SIAM J. COMPUT.
Vol. 32, No. 1, pp. 1–47

BLACK-BOX CONCURRENT ZERO-KNOWLEDGE REQUIRES (ALMOST) LOGARITHMICALLY MANY ROUNDS*

RAN CANETTI†, JOE KILIAN‡, EREZ PETRANK§, AND ALON ROSEN¶

Abstract. We show that any concurrent zero-knowledge protocol for a nontrivial language (i.e., for a language outside $\mathcal{BPP}$), whose security is proven via black-box simulation, must use at least $\tilde{\Omega}(\log n)$ rounds of interaction. This result achieves a substantial improvement over previous lower bounds and is the first bound to rule out the possibility of constant-round concurrent zero-knowledge when proven via black-box simulation. Furthermore, the bound is polynomially related to the number of rounds in the best known concurrent zero-knowledge protocol for languages in $\mathcal{NP}$ (which is established via black-box simulation).

Key words. cryptography, interactive protocols, zero knowledge, concurrent zero knowledge, round complexity, lower bounds

AMS subject classifications. 94A60, 68Q25, 68Q85, 68Q99

PII. S0097539701392949

1. Introduction.

Zero-knowledge proof-systems, introduced by Goldwasser, Micali, and Rackoff [21], are efficient interactive proofs that have the remarkable property of yielding nothing beyond the validity of the assertion being proved. The generality of zero-knowledge proofs has been demonstrated by Goldreich, Micali, and Wigderson [19], who showed that every NP-statement can be proved in zero-knowledge provided that one-way functions exist [23, 27]. Since then, zero-knowledge proofs have turned out to be an extremely useful tool in the design of various cryptographic protocols.

The original setting in which zero-knowledge proofs were investigated consisted of a single prover and a verifier that executed only one instance of the protocol at a time. A more realistic setting, especially in the age of the internet, is one that allows the *concurrent* execution of zero-knowledge protocols. In the concurrent setting (see Feige [14] and the more extensive treatment by Dwork, Naor, and Sahai [12]), many protocols (sessions) are executed at the same time, involving many verifiers which may be talking with the same (or many) provers simultaneously. (The so-called parallel composition considered in [18, 15, 17, 6, 4] is merely a special case.) This setting presents the new risk of a coordinated attack in which an adversary controls many verifiers, interleaving the executions of the protocols and choosing verifiers' messages based on other partial executions of the protocol. Since it seems unrealistic (and certainly undesirable) for honest provers to coordinate their actions so that zero-

*Received by the editors July 25, 2001; accepted for publication (in revised form) July 7, 2002; published electronically November 19, 2002. An extended abstract has appeared in *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, ACM, New York, 2001, pp. 570–579 [8].

http://www.siam.org/journals/sicomp/32-1/39294.html

†IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598 (canetti@watson.ibm.com).

‡Yianilos Labs, 707 State Rd., Rt. 206, Suite 212, Princeton, NJ 08540 (joe@pnylab.com).

§Department of Computer Science, Technion—Israel Institute of Technology, Haifa 32000, Israel (erez@cs.technion.ac.il).

¶Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel (alon@wisdom.weizmann.ac.il). Part of this author's work was done while he was visiting the IBM T.J. Watson Research Center.

knowledge is preserved, we must assume that, in each prover-verifier pair, the prover acts independently.

Loosely speaking, a zero-knowledge proof is said to be *concurrent zero-knowledge* if it remains zero-knowledge even when executed in the concurrent setting. Recall that, in order to demonstrate that a certain protocol is zero-knowledge, it is required to demonstrate that the view of every probabilistic polynomial-time adversary interacting with the prover can be simulated by a probabilistic polynomial-time machine (also known as the *simulator*). In the concurrent setting, the verifiers' view may include multiple sessions running at the same time. Furthermore, the verifiers may have control over the scheduling of the messages in these sessions (i.e., the order in which the interleaved execution of these sessions should be conducted). As a consequence, the simulator's task in the concurrent setting becomes considerably more complicated. In particular, standard techniques, based on "rewinding the adversary," run into trouble.

1.1. Previous work. Constructing a "round-efficient" concurrent zero-knowledge protocol for all languages in $\mathcal{NP}$ or even nontrivial languages (outside of $\mathcal{BPP}$) seems to be a challenging task. Intuition on the difficulty of this problem is given in [12], where it was argued that, for a specific 4-round zero-knowledge protocol and a specific recursive scheduling of n sessions, the straightforward adaptation of the simulator to the concurrent setting requires time exponential in n. The first lower bound demonstrating the difficulty of concurrent zero-knowledge was given by Kilian, Petrank, and Rackoff [26], who showed, building on the techniques of Goldreich and Krawczyk [18], that, for every language outside $\mathcal{BPP}$, there is no 4-round protocol whose concurrent execution is simulatable in polynomial time by a *black-box simulator*. (A black-box simulator is a simulator that has only black-box access to the adversarial verifier. Essentially all previously known proofs of security of zero-knowledge protocols use black-box simulators. An exception is the protocol of [22], which uses a nonstandard assumption of a "non black-box" nature.) This lower bound was later improved by Rosen to seven rounds [29].

Indeed, even ignoring issues of round efficiency, it was not clear whether there exists a concurrent zero-knowledge protocol for nontrivial languages without modifying the underlying model. Richardson and Kilian [28] exhibited a family of concurrent zero-knowledge protocols (parameterized by the number of rounds) for all languages in $\mathcal{NP}$. Their original analysis showed how to simulate in polynomial time $n^{O(1)}$ concurrent sessions only when the number of rounds in the protocol is at least n^{ϵ} (for some arbitrary $\epsilon > 0$). This result has recently been substantially improved by Kilian and Petrank [25], who show that the Richardson–Kilian protocol remains concurrent zero-knowledge even if it has $O(g(n) \cdot \log^2 n)$ rounds, where $g(\cdot)$ is any nonconstant function (e.g., $g(n) = \log \log n$).

We note that previously there was a considerable gap between the known upper and lower bounds on the round complexity of concurrent zero-knowledge (i.e., [25, 29]): the best known protocol has $\tilde{O}(\log^2 n)$ rounds, whereas the lower bound necessitates 7 rounds (via black-box simulation).[1] In particular, the question consisting of whether constant-round concurrent zero-knowledge protocols exist has been open.

1.2. Our result. We substantially narrow the above gap by presenting a lower bound on the number of rounds required by concurrent zero-knowledge. We show that,

[1] $f(n) = \tilde{O}(h(n))$ if there exist constants $c_1, c_2 > 0$ so that, for all sufficiently large n, $f(n) \leq c_1 \cdot (\log h(n))^{c_2} \cdot h(n)$.

in the context of black-box concurrent zero-knowledge, $\tilde{\Omega}(\log n)$ rounds of interaction are essential for nontrivial proof-systems.[2] This bound is the first to rule out the possibility of constant-round concurrent zero-knowledge, when proven via black-box simulation. Furthermore, the bound is polynomially related to the number of rounds in the best-known concurrent zero-knowledge protocol for languages outside $\mathcal{BPP}$ [25]. Our main result is stated in the following theorem.

THEOREM 1.1. *Let $r : N \to N$ be a function so that $r(n) = o(\frac{\log n}{\log \log n})$. Suppose that $\langle P, V \rangle$ is an $r(\cdot)$-round proof-system for a language L (i.e., on input x, the number of messages exchanged is at most $r(|x|)$) and that concurrent executions of P can be simulated in polynomial time using black-box simulation. Then $L \in \mathcal{BPP}$. The theorem holds even if the proof-system is only computationally sound (with negligible soundness error) and the simulation is only computationally indistinguishable (from the actual executions).*

1.3. Techniques. The proof of Theorem 1.1 builds on the works of Goldreich and Krawczyk [18], Kilian, Petrank, and Rackoff [26], and Rosen [29]. On a very high level, the proof proceeds by constructing a specific concurrent schedule of sessions and demonstrating that a black-box simulator cannot successfully generate a simulated accepting transcript for this schedule unless it "rewinds" the verifier *many times.* The work spent on these rewindings will be superpolynomial unless the number of rounds used by the protocol obeys the bound, or $L \in \mathcal{BPP}$. While the general outline of the proof remains roughly the same as in [18, 26, 29], the actual schedule of sessions and its analysis are new. One main idea that, together with other ideas, enables the proof of the bound is to have the verifier *abort* sessions depending on the history of the interaction. A more detailed outline, presenting both the general structure and the new ideas in the proof, appears in section 3.

Remark. The concurrent schedule in our proof is fixed and known to everybody. As a consequence, Theorem 1.1 is actually stronger than stated. It will hold even if the simulator knows the schedule in advance (in particular, it knows the number of concurrent sessions) and even if the schedule of the messages does not change dynamically (as a function of the history of the interaction).

1.4. Conclusions and open problems.

1.4.1. Alternative models. The lower bound presented here draws severe limitations on the ability of black-box simulators to cope with the standard concurrent zero-knowledge setting and provides motivation to consider relaxations of and augmentations to the standard model. Indeed, several works have managed to "bypass" the difficulty in constructing concurrent zero-knowledge protocols by modifying the standard model in a number of ways. Dwork, Naor, and Sahai augment the communication model with assumptions on the maximum delay of messages and skews of local clocks of parties [12, 13]. Damgård uses a common random string [11], and Canetti et al. use a public registry file [7].

A different approach would be to try to achieve security properties that are weaker than zero-knowledge but are still useful. For example, Feige and Shamir consider the notion of *witness indistinguishability* [14, 15], which is preserved under concurrent composition.

[2] $f(n) = \tilde{\Omega}(h(n))$ if there exist constants $c_1, c_2 > 0$ so that, for all sufficiently large n, $f(n) \geq c_1 \cdot h(n)/(\log h(n))^{c_2}$.

1.4.2. Alternative simulation techniques. Loosely speaking, the only advantage that a black-box simulator may have over the honest prover is the ability to "rewind" the interaction and explore different execution paths before proceeding with the simulation (as its access to the verifier's strategy is restricted to the examination of input/output behavior). As we show in our proof, such a mode of operation (i.e., the necessity to rewind every session) is a major contributor to the hardness of simulating many concurrent sessions. It is thus natural to think that a simulator that deviates from this paradigm (i.e., is non black-box in the sense that it does not have to rewind the adversary in order to obtain a faithful simulation of the conversation) would essentially bypass the main problem that arises while trying to simulate many concurrent sessions.

Hada and Tanaka [22] have considered some weaker variants of zero-knowledge and have exhibited a three-round protocol for $\mathcal{NP}$ (whereas only $\mathcal{BPP}$ has three-round black-box zero-knowledge [18]). Their protocol was an example for a zero-knowledge protocol not proven secure via black-box simulation. Alas, their analysis was based in an essential way on a strong and highly nonstandard hardness assumption.

In a recent breakthrough result, Barak [2] constructs a constant-round protocol for all languages in $\mathcal{NP}$ whose zero-knowledge property is proved using a *non black-box* simulator. Such a method of simulation enables him to bypass our impossiblity result (as well as [18, 26, 29]) and to perform cryptographic tasks otherwise considered unachievable. In particular, for every (predetermined) polynomial $p(\cdot)$, there exists a version of Barak's protocol that preserves its zero-knowledge property even when it is executed $p(n)$ times concurrently (where n denotes the size of the common input). As we show in our work, this task is unachievable via black-box simulation (unless $\mathcal{NP} \subseteq \mathcal{BPP}$).

1.4.3. Open problems. At first glance, it seems that Barak's protocol completely resolves the question of whether constant-round concurrent zero-knowledge protocols exist. Taking a closer look, however, one notices that the (polynomial) number of concurrent sessions relative to which the protocol should be secure is determined *before* the protocol is specified. Moreover, it turns out that the messages in the protocol are required to be longer than the number of concurrent sessions. Thus, from both a theoretical and a practical point of view, Barak's protocol is still not satisfactory. What we would like to have is a *single* protocol that preserves its zero-knowledge property even when it is executed concurrently for *any* (not predetermined) polynomial number of times. Such a property is indeed satisfied by the protocols of [28, 25] (alas, these protocols are not constant-round). This leaves open the question of whether constant-round concurrent zero-knowledge protocols indeed exist for all languages in $\mathcal{NP}$.

2. Preliminaries.

2.1. Probabilistic notation. Denote by $x \stackrel{R}{\leftarrow} X$ the process of uniformly choosing an element x in a set X. If $B(\cdot)$ is an event depending on the choice of $x \stackrel{R}{\leftarrow} X$, then $\Pr_{x \leftarrow X}[B(x)]$ (alternatively, $\Pr_x[B(x)]$) denotes the probability that $B(x)$ holds when x is chosen with probability $1/|X|$. Namely,

$$\Pr_{x \leftarrow X}[B(x)] = \sum_x \frac{1}{|X|} \cdot \chi(B(x)),$$

where χ is an indicator function so that $\chi(B) = 1$ if event B holds and equals zero otherwise. This notation extends in the natural way for events $B(\cdot, \ldots, \cdot)$ that depend

on k variables $x_1, x_2, \ldots, x_k$ that are uniformly chosen in k (possibly different) sets $X_1, X_2, \ldots, X_k$. That is, we denote by $\Pr_{x_1,x_2,\ldots,x_k}[B(x_1, x_2, \ldots, x_k)]$ the probability that $B(x_1, x_2, \ldots, x_k)$ holds when $x_1, x_2, \ldots, x_k$ are chosen with probability $1/(|X_1| \cdot |X_2| \cdots |X_k|)$.

2.2. Interactive proofs. We use the standard definitions of interactive proofs (interactive Turing machines) [21, 16] and arguments (also known as *computationally sound* proofs) [5]. Given a pair of interactive Turing machines, P and V, we denote by $\langle P, V\rangle(x)$ the random variable representing the (local) output of V when interacting with machine P on common input x, when the random input to each machine is uniformly and independently chosen. We consider interactive proof-systems in which the soundness error is negligible. The term *negligible* is used for denoting functions that are (asymptotically) smaller than one over any polynomial. More precisely, a function $\nu(\cdot)$ from nonnegative integers to reals is called *negligible* if, for every constant $c > 0$ and all sufficiently large n, it holds that $\nu(n) < n^{-c}$.

DEFINITION 2.1 (interactive proof-system). *A pair of interactive machines $\langle P, V\rangle$ is called an* interactive proof-system *for a language L if machine V is polynomial-time and the following two conditions hold with respect to some negligible function $\nu(\cdot)$:*

- Completeness. *For every $x \in L$,*

$$\Pr[\langle P, V\rangle(x) = 1] \geq 1 - \nu(|x|).$$

- Soundness. *For every $x \notin L$ and every interactive machine B,*

$$\Pr[\langle B, V\rangle(x) = 1] \leq \nu(|x|).$$

Definition 2.1 can be relaxed to require only the soundness error that is bounded away from $1 - \nu(|x|)$. This is so since the soundness error can always be made negligible by sufficiently many parallel repetitions of the protocol (as such may occur anyhow in the concurrent model). However, we do not know whether this condition can be relaxed in the case of computationally sound proofs (i.e., when the soundness condition is required to hold only for machines B that are implementable by polysize circuits). In particular, in this case, parallel repetitions do not necessarily reduce the soundness error (cf. [3]).

2.3. Concurrent zero-knowledge. Let $\langle P, V\rangle$ be an interactive proof for a language L, and consider a *concurrent adversary* (verifier) V^* that, given input $x \in L$, interacts with an unbounded number of independent copies of P (all on common input x). The concurrent adversary V^* is allowed to interact with the various copies of P concurrently, without any restrictions over the scheduling of the messages in the different interactions with P. (In particular, V^* has control over the scheduling of the messages in these interactions.) The *transcript* of a concurrent interaction consists of the common input x followed by the sequence of prover and verifier messages exchanged during the interaction. We denote by $\text{view}^P_{V^*}(x)$ a random variable describing the content of the random tape of V^* and the transcript of the concurrent interaction between P and V^* (that is, all messages that V^* sends and receives during the concurrent interactions with P, on common input x).

Remark. The actual definition of concurrent zero-knowledge requires that the concurrent adversary V^* explicitly specifies to which session the next scheduled message belongs. However, in the proof of Theorem 1.1, we consider a "weaker" concurrent

adversary V^* that is running only a fixed scheduling of sessions (and so does not determine the schedule dynamically). In particular, there will be no need to use a formalism for specifying to which session the next scheduled message belongs.

DEFINITION 2.2 (concurrent zero-knowledge). *Let $\langle P, V\rangle$ be an interactive proof-system for a language L. We say that $\langle P, V\rangle$ is* concurrent zero-knowledge *if, for every polynomial-time concurrent adversary V^*, there exists a probabilistic polynomial-time algorithm S_{V^*} such that the ensembles $\{\text{view}^P_{V^*}(x)\}_{x\in L}$ and $\{S_{V^*}(x)\}_{x\in L}$ are computationally indistinguishable.*

2.4. Black-box concurrent zero-knowledge. Loosely speaking, the definition of black-box zero-knowledge requires that there exists a "universal" simulator, S, so that, for every $x \in L$ and every probabilistic polynomial-time adversary V^*, the simulator S produces a distribution that is indistinguishable from $\text{view}^P_{V^*}(x)$ while using V^* as an oracle (i.e., in a "black-box" manner). We assume concurrent adversaries V^* are modeled by polysized circuits (capturing nonuniform, deterministic verifiers viewed as an oracle; cf. [18, 16, 26]).

Before we proceed with the formal definition, we will have to overcome a technical difficulty arising from an inherent difference between the concurrent setting and the "stand-alone" setting. In "stand-alone" zero-knowledge, the length of the output of the simulator depends only on the protocol and the size of the common input x. It is thus reasonable to require that the simulator run in time that depends only on the size of x, regardless of the running time of its black-box. However, in black-box concurrent zero-knowledge, the output of the simulator is an entire schedule, and its length depends on the running time of the concurrent adversary. Therefore, if we naively require that the running time of the simulator be a fixed polynomial in the size of x, then we end up with an unsatisfiable definition. (As for any simulator S, there is an adversary V^* that generates a transcript that is longer than the running time of S.)

One way to solve the above problem is to have for *each* fixed polynomial $q(\cdot)$ a simulator S_q that simulates "only" all $q(\cdot)$-sized circuits V^*. Clearly, the running time of the simulator now depends on the running time of V^* (which is an upper bound on the size of the schedule), and the above problem does not occur anymore. Another (more restrictive) way to overcome the above problem would be to consider a simulator S_q that simulates "only" all adversaries V^* which run at most $q(|x|)$ sessions during their execution. (We stress that $q(\cdot)$ is chosen *after* the protocol is determined.) Such simulators should run in worst-case time that is a fixed polynomial in $q(|x|)$ and in the size of the common input x. (Note that by letting S_q "know" $q(\cdot)$ in advance, we actually *strengthen* the lower bound.) In what follows, we choose to adopt the latter formalization. We stress that both formalizations are general enough to include all *known* black-box zero-knowledge proofs.

DEFINITION 2.3 (black-box concurrent zero-knowledge). *Let $\langle P, V\rangle$ be an interactive proof-system for a language L. We say that $\langle P, V\rangle$ is black-box concurrent zero-knowledge if, for every polynomial $q(\cdot)$, there exists a probabilistic polynomial-time[3] algorithm S_q so that, for every concurrent adversary circuit V^* that runs at most $q(|x|)$ concurrent sessions, $S_q(x)$ runs in time polynomial in $q(|x|)$ and $|x|$ and satisfies that the ensembles $\{\text{view}^P_{V^*}(x)\}_{x\in L}$ and $\{S_q(x)\}_{x\in L}$ are computationally indistinguishable.*

[3]See below for a discussion on expected versus strict probabilistic polynomial time.

2.5. Additional conventions.

Deviation gap and expected polynomial-time simulators. The *deviation gap* of a simulator S for a proof-system $\langle P, V \rangle$ is defined, somewhat informally, as follows. Consider a distinguisher D that is required to decide whether its input consists of $\text{view}^P_{V^*}(x)$ or the transcript that was produced by S. The deviation gap of D is the difference between the probability that D outputs 1 given an output of S and the probability that D outputs 1 given $\text{view}^P_{V^*}(x)$. The deviation gap of S is the deviation gap of the best polynomial-time distinguisher D. In our definitions of concurrent zero-knowledge (Definitions 2.2 and 2.3), the deviation gap of the simulator is required to be negligible in $|x|$.

For our lower bound, we allow simulators that run in strict (worst-case) polynomial time and have a deviation gap of at most 1/4. As for expected polynomial-time simulators, one can use a standard argument to show that any simulator running in expected polynomial time and having a deviation gap of at most 1/8 can be transformed into a simulator that runs in strict (worst-case) polynomial time and has a deviation gap of at most 1/4. In particular, our lower bound (on simulators that run in strict polynomial time and have a deviation gap of at most 1/4) extends to a lower bound on simulators running in expected polynomial time (and having a deviation gap of as large as 1/8).

Query conventions. By k-round protocols, we mean protocols in which $2k + 2$ messages are exchanged subject to the following conventions. The first message is a fixed initiation message by the verifier, denoted by $\mathtt{v}_1$, which is answered by the prover's first message, denoted by $\mathtt{p}_1$. The following verifier and prover messages are denoted by $\mathtt{v}_2, \mathtt{p}_2, \ldots, \mathtt{v}_{k+1}, \mathtt{p}_{k+1}$, where $\mathtt{v}_{k+1}$ is an `ACCEPT/REJECT` message indicating whether the verifier has accepted its input, and the last message (i.e., $\mathtt{p}_{k+1}$) is a fixed acknowledgment message sent by the prover.[4] Clearly, any protocol in which $2k$ messages are exchanged can be modified to fit this form (by adding at most two messages).

We impose the following technical restrictions on the simulator (but claim that each of these restrictions can be easily satisfied): As in [18], the queries of the simulator are prefixes of possible execution transcripts (in the concurrent setting).[5] Such a prefix is a sequence of alternating prover and verifier messages (which may belong to different sessions as determined by the fixed schedule) that ends with a prover message. The answer to the queries made by the simulator consists of a single verifier message (which belongs to the next scheduled session). We assume that the simulator never repeats the same query twice. In addition, we assume that, before making a query $\overline{q} = (b_1, a_1, \ldots, b_t, a_t)$, where the a's are prover messages, the simulator has made queries to all relevant prefixes (i.e., $(b_1, a_1, \ldots, b_i, a_i)$ for every $i < t$) and has obtained the b_i's as answers. Finally, we assume that, before producing output $(b_1, a_1, \ldots, b_T, a_T)$, the simulator makes the query $(b_1, a_1, \ldots, b_T, a_T)$.

3. Proof outline. This section contains an outline of the proof of Theorem 1.1. The actual proof will be given in sections 4 and 5. To facilitate reading, we partition the outline into two parts: The first part reviews the general framework. (This part mainly follows previous works, namely, [17, 26, 29].) The second part concentrates on the actual schedule and the specifics of our lower bound argument.

[4] The $\mathtt{p}_{k+1}$ message is an artificial message included in order to "streamline" the description of the adversarial schedule. (The schedule will be defined in section 4.1.1.)

[5] For the sake of simplicity, we choose to omit the input x from the transcript's representation (as it is implicit in the description of the verifier anyway).

3.1. The high-level framework. Consider a k-round concurrent zero knowledge proof-system $\langle P, V\rangle$ for language L, and let S be a black-box simulator for $\langle P, V\rangle$. We use S to construct a $\mathcal{BPP}$ decision procedure for L. For this purpose, we construct a family $\{V_h\}$ of "cheating verifiers." To decide on an input x, run S with a cheating verifier V_h that was chosen at random from the constructed family, and decide that $x \in L$ iff S outputs an accepting transcript of V_h.

The general structure of the family $\{V_h\}$ is roughly as follows. A member V_h in the family is identified via a hash function h taken from a hash-function family H having "much randomness" (or high independence). Specifically, the independence of H will be larger than the running time of S. This guarantees that, for our purposes, a function drawn randomly from H behaves like a random function. We define some fixed concurrent schedule of a number of sessions between V_h and the prover. In each session, V_h runs the code of the honest verifier V on input x and random input $h(a)$, where a is the current history of the (*multisession*) interaction at the point where the session starts. V_h accepts if all of the copies of V accept.

The proof of validity of the decision procedure is structured as follows. Say that S *succeeds* if it outputs an accepting transcript of V_h. It is first claimed that, if $x \in L$, then a valid simulator S must succeed with high probability. Roughly speaking, this is so because each session behaves like the original proof-system $\langle P, V\rangle$, and $\langle P, V\rangle$ accepts x with high probability, demonstrating that the simulator almost never succeeds when $x \notin L$ is much more involved. Given S, we construct a "cheating prover" P^* that makes the honest verifier V accept x with probability that is polynomially related to the success probability of S. The soundness of $\langle P, V\rangle$ now implies that, in this case, S succeeds only with negligible probability. See the details below.

3.1.1. Session-prefixes and useful session-prefixes. In order to complete the high-level description of the proof, we must first define the following notions that play a central role in the analysis. Consider the conversation between V_h and a prover. A *session-prefix* a is a prefix of this conversation that ends at the point where some new session starts (including the first verifier message in that session). (Recall that V's random input for that new session is set to $h(a)$.) Next, consider the conversation between S and V_h in some run of S. (Such a conversation may contain many interleaved and incomplete conversations of V_h with a prover.) Roughly speaking, a message sent by S to the simulated V_h is said to have a session-prefix a if it relates to the session where the verifier randomness is $h(a)$. A session-prefix a is called *useful* in a run of S if the following hold.

1. It was accepted (i.e., V_h sent an `ACCEPT` message for session-prefix a).
2. V_h has sent exactly $k+1$ messages for session-prefix a.

Loosely speaking, condition 2 implies that S did not rewind the relevant session-prefix, where "rewind session-prefix a" is an informal term meaning that S rewinds V_h to a point where V_h provides a second continuation for session-prefix a. By rewinding session-prefix a, the simulator is able to obtain more than $k+1$ verifier messages for session-prefix a. This is in contrast to an actual execution of the protocol $\langle P, V\rangle$, in which V sends exactly $k+1$ messages.

3.1.2. The construction of the cheating prover. Using the above terms, we sketch the construction of the cheating prover P^*. It first randomly chooses a function $h \stackrel{R}{\leftarrow} H$ and an index (of a session-prefix) i. It then emulates an interaction between S and V_h, with the exception that P^* uses the messages sent by S that have the ith session-prefix as the messages that P^* sends to the actual verifier it interacts

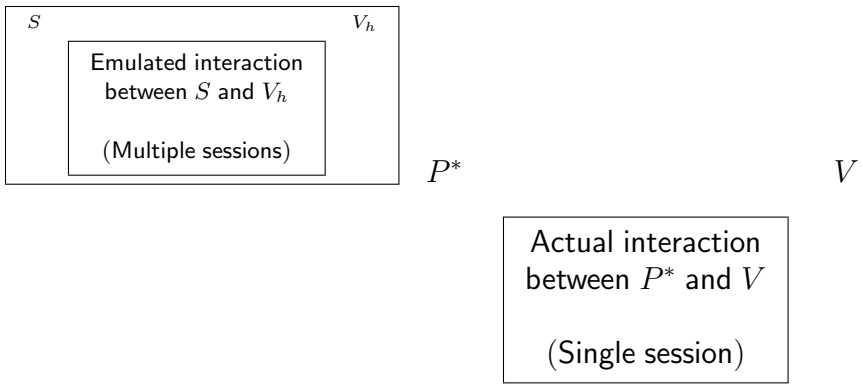


FIG. 3.1. *The above describes the strategy of the cheating prover P^*. The box on the left-hand side represents the (multiple session) emulation of the interaction between S and V_h (executed "internally" by P^*). The box on the right-hand side represents the actual execution of a single session between P^* and V. (Recall that P^* relays some of the actual interaction messages to its internal emulation.)*

with; similarly, it uses the messages received from the actual verifier V instead of V_h's messages in the ith session-prefix. The strategy of the cheating prover is depicted in Figure 3.1.

3.1.3. The success probability of the cheating prover. We next claim that, if the session-prefix chosen by P^* is useful, then $\langle P^*, V\rangle(x)$ accepts. The key point is that, whenever P^* chooses a useful session-prefix, the following two conditions (corresponding to the two conditions in the definition of a useful session-prefix) are satisfied:

1. The session corresponding to the ith session-prefix is accepted by V_h (and so by V).
2. P^* manages to reach the end of the $\langle P^*, V\rangle$ interaction without "getting into trouble."[6]

Loosely speaking, item 1 is implied by condition 1 in the definition of a useful session-prefix. As for item 2, this just follows from the fact that S does not rewind the ith session-prefix (as implied by condition 2 in the definition of a useful session-prefix). In particular, P^* (playing the role of V_h) will not have to send the jth verifier message with the ith session-prefix more than once to S (since the number of messages sent by V_h for that session-prefix is exactly $k+1$).

Since the number of session-prefixes in an execution of S is bounded by a polynomial, it follows that, if the conversation between S and V_h contains a useful session-prefix with nonnegligible probability, then $\langle P^*, V\rangle(x)$ accepts with nonnegligible probability.

3.2. The schedule and additional ideas. Using the above framework, the crux of the lower bound is to come up with a schedule and V_h's that allow us to demonstrate that, whenever S succeeds, the conversation between S and V_h contains

[6]The problem is that P^* does not know V's random coins, and so it cannot compute the verifier's answers by himself. Thus, whenever P^* is required in the emulation to send the jth verifier message in the protocol more than once to S, it might get into trouble (since it gets the jth verifier message only once from V).

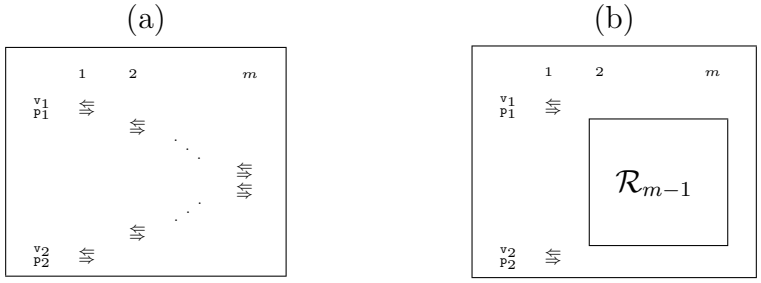


FIG. 3.2. *The "telescopic" schedule used by* [26] *to demonstrate the impossibility of black-box concurrent zero-knowledge in* 2 *rounds. Columns correspond to* n *individual sessions, and rows correspond to the time progression.* (a) *depicts the schedule explicitly.* (b) *depicts the schedule in a recursive manner. (*$\mathcal{R}_m$ *denotes the recursive schedule for* m *sessions.)*

a useful session-prefix. (As we have argued above, it is in fact sufficient that the conversation between S and V_h contains a useful session-prefix with nonnegligible probability.) This is done next.

3.2.1. The 2-round case. Our starting point is the schedule used in [26] to demonstrate the impossibility of black-box concurrent zero-knowledge with protocols in which 4 messages are exchanged (i.e., $\mathtt{v}_1, \mathtt{p}_1, \mathtt{v}_2, \mathtt{p}_2$). The schedule is recursive and consists of n concurrent sessions. (n is polynomially related to the security parameter.) Given parameter $m \leq n$, the scheduling on m sessions (denoted $\mathcal{R}_m$) proceeds as follows (see Figure 3.2 for a graphical description):

1. If $m = 1$, the relevant session exchanges all of its messages (i.e., $\mathtt{v}_1, \mathtt{p}_1, \mathtt{v}_2, \mathtt{p}_2$).
2. Otherwise (i.e., if $m > 1$):
   Initial message exchange. The first session (out of m sessions) exchanges 2 messages (i.e., messages $\mathtt{v}_1, \mathtt{p}_1$);
   Recursive call. The schedule is applied recursively on the remaining $m - 1$ sessions;
   Final message exchange. The first session (out of m sessions) exchanges 2 messages (i.e., messages $\mathtt{v}_2, \mathtt{p}_2$).

At the end of each session, V_h continues in the interaction iff the transcript of the session that has just terminated would have been accepted by the prescribed verifier V. This means that, in order to proceed beyond the ending point of the ℓth session, the simulator must make the honest verifier accept the sth session for all $s > \ell$.

Suppose now that S succeeds in simulating the above V_h, but the conversation between S and V_h does not contain a useful session-prefix. Since V_h proceeds beyond the ending point of a session only if this session is accepted, then the only reason for which the corresponding session-prefix can be nonuseful is because S has rewound that session-prefix. In other words, a session-prefix becomes nonuseful iff S resends the first prover message in the protocol (i.e., $\mathtt{p}_1$).[7] This should cause V_h to resend the second verifier message (i.e., $\mathtt{v}_2$), thus violating condition 2 in the definition of a useful session-prefix (see section 3.1.1).

[7]Notice that the first prover message in the protocol (i.e., $\mathtt{p}_1$) is the only place in which rewinding the interaction may cause a session-prefix to be nonuseful. The reason for this is that the first verifier message in the protocol (i.e., $\mathtt{v}_1$) is part of the session-prefix. Rewinding past this message (i.e., $\mathtt{v}_1$) would modify the session-prefix itself. As for $\mathtt{p}_2$, it is clear that rewinding this message would not cause any change in verifier messages that correspond to the relevant session-prefix (since $\mathtt{v}_1$ and $\mathtt{v}_2$ occur before $\mathtt{p}_2$ anyway).

The key observation is that, whenever the first prover message in the ℓth session is modified, so is the session-prefix of the sth session for *all* $s > \ell$. Thus, whenever S resends the first prover message in the ℓth session, it must do so also in the sth session for all $s > \ell$ (since otherwise the "fresh" session-prefix of the sth session that is induced by resending the above message will be useful). However, this means that the work $W(m)$, invested in the simulation of a schedule with m levels, must satisfy $W(m) \geq 2 \cdot W(m-1)$ for all m. Thus either the conversation between V_h and S contains a useful session-prefix (in which case we are done), or the simulation requires exponential time (since $W(m) \geq 2 \cdot W(m-1)$ solves to $W(n) \geq 2^{n-1}$).

3.2.2. The *k*-round case—first attempt. The case of k rounds may proceed as follows. Given the parameter $m \leq n$ (denoting the number of sessions in $\mathcal{R}_m$), do:

1. If $m = 1$, the relevant session exchanges all of its messages (i.e., messages $\mathtt{v}_1, \mathtt{p}_1, \ldots, \mathtt{v}_{k+1}, \mathtt{p}_{k+1}$).
2. Otherwise, for $j = 1, \ldots, k+1$:
   Message exchange. The first session (out of m sessions) exchanges two messages (i.e., $\mathtt{v}_j, \mathtt{p}_j$);
   Recursive call. If $j < k+1$, the scheduling is applied recursively on $\lfloor \frac{m-1}{k} \rfloor$ new sessions.
   (This is done using the next $\lfloor \frac{m-1}{k} \rfloor$ remaining sessions out of $2, \ldots, m$.)

As before, at the end of each session, V_h continues in the interaction iff the transcript of the session that has just terminated would have been accepted by the prescribed verifier V. The schedule is depicted in Figure 3.3.

The crucial problem of the above schedule is that one can come up with a k-round protocol and a corresponding simulator that manages to successfully simulate V_h *and*

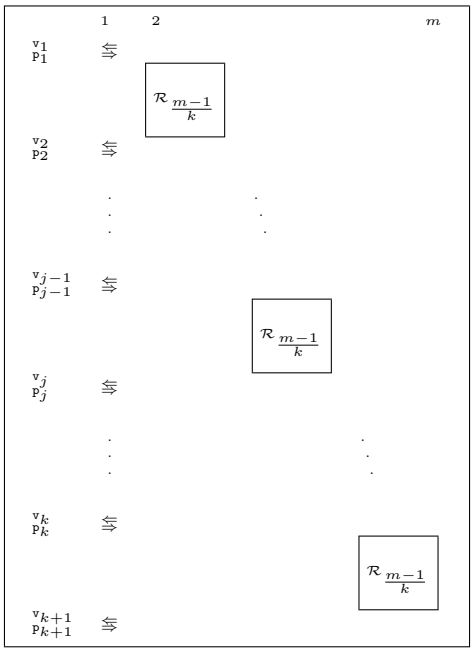


FIG. 3.3. *First attempt to generalize the recursive schedule ($\mathcal{R}_m$ with m sessions) for k-round protocols. Columns correspond to m individual sessions, and rows correspond to the time progression.*

cause all session-prefixes in its conversation with V_h to be nonuseful. Specifically, there exist protocols (cf. [28]) in which the simulator is required to successfully rewind an honestly behaving verifier exactly once for every session. Whereas, in the case of 2 rounds, this could have had devastating consequences (since, in the case of the previous schedule, it would have implied $W(m) \geq (k+1) \cdot W(m-1) = 2 \cdot W(m-1)$, which solves to $W(n) \geq 2^{n-1}$) in the general case (i.e., when $k+1 > 2$) when any rewinding of the schedule that we have suggested would have forced the simulator to reinvest simulation "work" only for $\frac{m-1}{k}$ sessions. Note that such a simulator satisfies $W(m) = (k+1) \cdot W(\frac{m-1}{k})$, which solves to $k^{O(\log_k n)} = n^{O(1)}$. In particular, by investing a polynomial amount of work, the simulator is able to make all session-prefixes not useful while successfully simulating all sessions.

3.2.3. The *k*-round case—second attempt. One method of circumventing this difficulty was used in [29]. However, that method extends the lower bound only up to 3 rounds (more precisely, 7 messages). Here we use a different method. What we do is let the cheating verifier abort (i.e., refuse to answer) every message in the schedule with some predetermined probability (independently of other messages). To do this, we first add another, binary hash function, g, to the specification of V_h. This hash function is taken from a family G with sufficient independence so that it looks like a random binary function. Now, before generating the next message in some session, $V_{g,h}$ first applies g to some predetermined part of the conversation so far. If g returns 0, then $V_{g,h}$ aborts the session by sending an `ABORT` message. If g returns 1, then $V_{g,h}$ is run as usual.

The rationale behind the use of aborts can be explained as follows. Recall that a session-prefix a stops being useful only when $V_{g,h}$ sends more than k messages whose session-prefix is a. This means that a stops being useful only if S rewinds the session-prefix a *and, in addition,* g returns 1 in at least two of the continuations of a. This means that S is expected to rewind session-prefix a several times before it stops being useful. Since each rewinding of a involves extra work of S on higher-level sessions, this may force S to invest considerably more work before a session stops being useful.

A bit more specifically, let p denote the probability, taken over the choice of g, that g returns 1 on a given input. In each attempt, the session is not aborted with probability p. Thus S is expected to rewind a session prefix $1/p$ times before it becomes nonuseful. This gives us hope that, in order to make sure that no session-prefix is useful, S must do work that satisfies a condition of the following sort:

$$W(m) \geq \Omega(1/p) \cdot W\left(\tfrac{m-1}{k}\right). \tag{3.1}$$

This would mean that the work required to successfully simulate n sessions *and* make all session-prefixes nonuseful is at least $\Omega(p^{-\log_k n})$. Consequently, when the expression $p^{-\log_k n}$ is superpolynomial, there is hope that the conversation between S and V_h contains a useful session-prefix with nonnegligible probability.

3.2.4. The *k*-round case—final version. However, demonstrating (3.1) brings up the following difficulty. Once the verifier starts aborting sessions, the probability that a session is ever completed may become too small. As a consequence, it is not clear anymore that the simulator must invest simulation "work" for all sessions in the schedule. It may very well be the case that the simulator will go about the simulation task while "avoiding" part of the simulation "work" in some recursive invocations (as some of these invocations may be aborted anyway during the simulation). In other

words, there is no guarantee that the recursive "work" invested by the simulator behaves like (3.1).[8]

To overcome this problem, we replace each session in the above schedule with a "block" of, say, n sessions (see Figure 4.1). We now have n^2 sessions in a schedule. (This choice of parameters is arbitrary and is made for convenience of presentation.) The modified $V_{g,h}$ accepts a block of n sessions if at least $1/2$ of the nonaborted sessions in this block were accepted and not too many of the sessions in this block were aborted. Once a block is rejected, $V_{g,h}$ halts. At the end of the execution, $V_{g,h}$ accepts if all blocks were accepted. The above modification guarantees that, with a careful setting of the parameters, the simulator's recursive "work" must satisfy (3.1) at least with overwhelming probability.

3.2.5. Setting the value of p. It now remains to set the value of p so that the simuatior's work behaves as in (3.1). Clearly, the smaller p is chosen to be, the larger $p^{-\log_k n}$ is. However, p cannot be too small, or else the probability that a session will ever be completed will be too small, and condition 1 in the definition of a useful session-prefix (section 3.1.1) will not be satisfied. Specifically, a k-round protocol is completed with probability p^k. We thus have to make sure that p^k is not negligible (and, furthermore, that $p^k \cdot n \gg 1$).

In the proof, we set $p = n^{-1/2k}$. This will guarantee that a session is completed with probability $p^k = n^{-1/2}$. (Thus condition 1 will hopefully be satisfied.) Furthermore, since $p^{-\log_k n}$ is superpolynomial whenever $k = o(\log n/\log\log n)$, there is hope that condition 2 in the definition of a useful session-prefix (section 3.1.1) will be satisfied for $k = o(\log n/\log\log n)$.

3.3. The actual analysis. Demonstrating that there exist many accepted session-prefixes is straightforward. Demonstrating that one of these session-prefixes is useful requires arguing on the dependency between the expected work done by the simulator and its success probability. This is a tricky business since the choices made by the simulator (and, in particular, the amount of effort spent on making each session nonuseful) may depend on past events.

We go about this task by pinpointing a special (combinatorial) property that holds for *any* successful run of the simulator, unless the simulator runs in superpolynomial time (Lemma 5.9). Essentially, this property states that there exists a block of sessions such that none of the session-prefixes in this block were rewound too many times. Using this property, we show (in Lemma 5.7) that the probability (over the choices of $V_{g,h}$ and the simulator) that a run of the simulator contains no useful session-prefix is negligible.

4. The actual proof (of Theorem 1.1). Assuming toward the contradiction that a black-box simulator, denoted S, contradicting Theorem 1.1 exists, we will describe a probabilistic polynomial-time decision procedure for L based on S. The first step toward describing the decision procedure for L involves the construction of an adversarial verifier in the concurrent model. This is done next.

4.1. The concurrent adversarial verifier. The description of the adversarial strategy proceeds in several steps. We start by describing the underlying fixed sched-

[8] To see this, imagine that the value of p is set to some negligible function (in n). In such a case, with overwhelming probability, *all* sessions in the schedule are aborted by $V_{g,h}$. However, this means that the distribution of interactions of P with $V_{g,h}$ can be easily simulated by a simulator that "does nothing." All that it has to do in order to produce a faithful simulation (at least with overwhelming probability) is output a transcript in which all sessions in the schedule are aborted.

ule of messages. Once the schedule is presented, we describe the adversary's strategy regarding the contents of the verifier messages.

4.1.1. The schedule. For each $x \in \{0,1\}^n$, we consider the following concurrent scheduling of n^2 sessions, all run on common input x.[9] The scheduling is defined recursively, where the scheduling of $m \leq n^2$ sessions (denoted $\mathcal{R}_m$) proceeds as follows:[10]

1. If $m \leq n$, sessions $1, \ldots, m$ are executed sequentially until they are all completed;
2. Otherwise, for $j = 1, \ldots, k+1$:
   Message exchange. Each of the first n sessions exchanges two messages (i.e., $\mathtt{v}_j, \mathtt{p}_j$);
   (These first n sessions out of $\{1, \ldots, m\}$ will be referred to as the *main sessions* of $\mathcal{R}_m$.)
   Recursive call. If $j < k+1$, the scheduling is applied recursively on $\lfloor \frac{m-n}{k} \rfloor$ new sessions.
   (This is done using the next $\lfloor \frac{m-n}{k} \rfloor$ remaining sessions out of $1, \ldots, m$.)

The schedule is depicted in Figure 4.1. We stress that the verifier typically postpones its answer (i.e., $\mathtt{v}_j$) to the last prover's message (i.e., $\mathtt{p}_{j-1}$) until after a recursive subschedule is executed and that, in the jth iteration of step 2, $\lfloor \frac{m-n}{k} \rfloor$ new sessions are initiated (with the exception of the first iteration, in which the first n (main) sessions are initiated as well). The order in which the messages of various sessions are exchanged (in the first part of step 2) is fixed but immaterial. Say that we let the first session proceed, and then the second, and so on. That is, we have the order $\mathtt{v}_j^{(1)}, \mathtt{p}_j^{(1)}, \ldots, \mathtt{v}_j^{(n)}, \mathtt{p}_j^{(n)}$, where $\mathtt{v}_j^{(i)}$ (resp., $\mathtt{p}_j^{(i)}$) denotes the verifier's (resp., prover's) jth message in the ith session.

The set of n sessions that are explicitly executed during the message exchange phase of the recursive invocation (i.e., the main sessions) is called a *recursive block*. (Notice that each recursive block corresponds to exactly one recursive invocation of the schedule.) Taking a closer look at the schedule, we observe that every session in the schedule is explicitly executed in exactly one recursive invocation (that is, belongs to exactly one recursive block). Since the total number of sessions in the schedule is n^2, and since the message exchange phase in each recursive invocation involves the explicit execution of n sessions (in other words, the size of each recursive block is n), we have that the total number of recursive blocks in the schedule equals n. Since each recursive invocation of the schedule involves the invocation of k additional subschedules, the recursion actually corresponds to a k-ary tree with n nodes. The depth of the recursion is thus $\lfloor \log_k((k-1)n+1) \rfloor$, and the number of "leaves" in the recursion (i.e., subschedules of size at most n) is at least $\lfloor \frac{(k-1)n+1}{k} \rfloor$.

Identifying sessions according to their recursive block. To simplify the exposition of the proof, it will be convenient to associate every session appearing in the schedule with a pair of indices $(\ell, i) \in \{1, \ldots, n\} \times \{1, \ldots, n\}$ rather than with a single index $s \in \{1, \ldots, n^2\}$. The value of $\ell = \ell(s) \in \{1, \ldots, n\}$ will represent the index of the recursive block to which session s belongs (according to some canonical enumeration of the n invocations in the recursive schedule, say, according to the order in which

[9] Recall that each session consists of $2k+2$ messages, where $k \stackrel{\text{def}}{=} k(n) = o(\log n / \log \log n)$.

[10] In general, we may want to define a recursive scheduling for sessions $i_1, \ldots, i_m$ and denote it by $\mathcal{R}_{i_1,\ldots,i_m}$ (see Appendix A for a more formal description of the schedule). We choose to simplify the exposition by renaming these sessions as $1, \ldots, m$, and we denote the scheduling by $\mathcal{R}_m$.

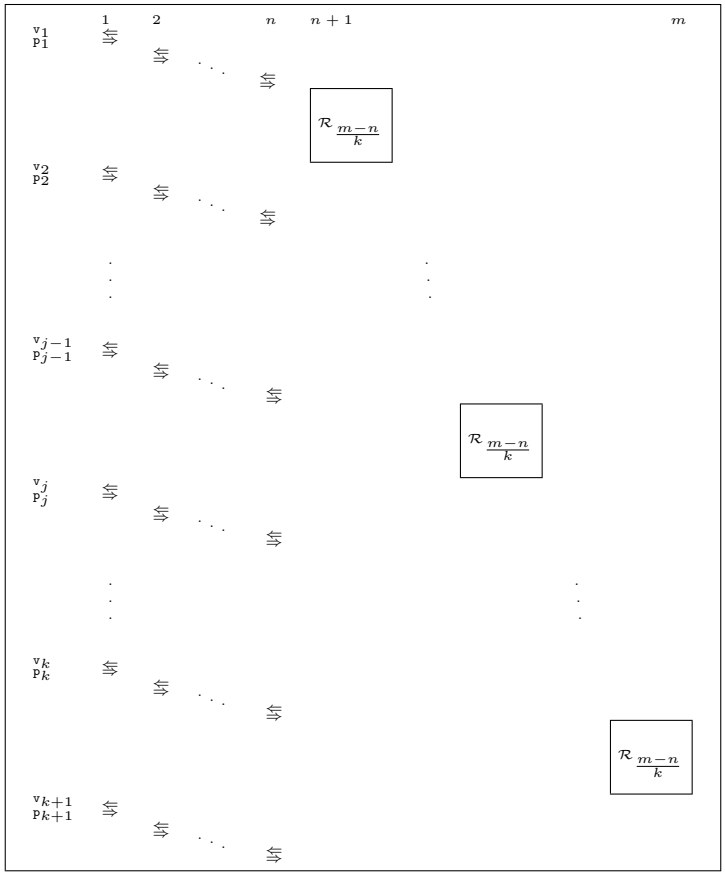


FIG. 4.1. *The recursive schedule $\mathcal{R}_m$ for m sessions. Columns correspond to m individual sessions, and rows correspond to the time progression.*

they are invoked), whereas the value of $i = i(s) \in \{1, \ldots, n\}$ will represent the index of session s within the n sessions that belong to the ℓth recursive block. (In other words, session (ℓ, i) is the ith main session of the ℓth recursive invocation in the schedule.) Typically, when we explicitly refer to messages of session (ℓ, i), the index of the corresponding recursive block (i.e., ℓ) is easily deducible from the context. In such cases, we will sometimes omit the index ℓ from the "natural" notation $\mathtt{v}_j^{(\ell,i)}$ (resp., $\mathtt{p}_j^{(\ell,i)}$) and stick to the notation $\mathtt{v}_j^{(i)}$ (resp., $\mathtt{p}_j^{(i)}$). Note that the values of (ℓ, i) and the session index s are completely interchangeable (in particular, $\ell = s \mathsf{\ div\ } n$ and $i = s \mathsf{\ mod\ } n$).

DEFINITION 4.1 (identifiers of the next message). *The schedule defines a mapping from partial execution transcripts ending with a prover message to the* identifiers of the next verifier message*—that is, the session and round number to which the next verifier message belongs.* (Recall that such partial execution transcripts correspond to queries of a black-box simulator, and so the mapping defines the identifier of the answer.) *For such a query $\overline{q} = (b_1, a_1, \ldots, b_t, a_t)$, we denote by $\pi_{\mathrm{sn}}(\overline{q}) = (\ell, i) \in \{1, \ldots, n\} \times \{1, \ldots, n\}$ the session to which the next verifier message belongs and by $\pi_{\mathrm{msg}}(\overline{q}) = j \in \{1, \ldots, k+1\}$ its index within the verifier's messages in this session.*

We stress that the identifiers of the next message are uniquely determined by the

number of messages appearing in the query (and are not affected by the contents of these messages).

4.1.2. Toward constructing an adversarial verifier. Once the identifiers of the next verifier message are deduced from the query's length, one has to specify a strategy according to which the contents of the next verifier message will be determined. Loosely speaking, our adversary verifier has two options: Either it will send the answer that would have been sent by an honest verifier (given the messages in the query that are relevant to the current session), or it will choose to deviate from the honest verifier strategy and abort the interaction in the current session. (This will be done by answering with a special `ABORT` message.) Since, in a nontrivial zero-knowledge proof-system, the honest verifier is always probabilistic (cf. [20]), and since the "abort behavior" of the adversary verifier should be "unpredictable" for the simulator, we have that both options require a source of randomness (either for computing the contents of the honest verifier answer or for deciding whether to abort the conversation). As is already customary in works of this sort [18, 26, 29], we let the source of randomness be a hash function with sufficiently high independence (which is "hard-wired" into the verifier's description), and we consider the execution of a black-box simulator that is given access to such a random verifier. (Recall that the simulator's queries correspond to partial execution transcripts and thus contain the whole history of the interaction so far.)

Determining the randomness for a session. Focusing (first) on the randomness required to compute the honest verifier's answers, we ask what the input of the above hash function should be. A naive solution would be to let the randomness for a session depend on the session's index. That is, to obtain randomness for session $(\ell, i) = \pi_{\mathrm{sn}}(\overline{q})$, apply the hash function on the value (ℓ, i). This solution will indeed imply that every two sessions have independent randomness (as the hash function will have different inputs). However, the solution seems to fail to capture the difficulty arising in the simulation (of multiple concurrent sessions). What we would like to have is a situation in which, whenever the simulator rewinds a session (that is, feeds the adversary verifier with a different query of the same length), it causes the randomness of some other session (say, one level down in the recursive schedule) to be completely modified. To achieve this, we must cause the randomness of a session to depend also on the history of the entire interaction. Changing even a single message in this history would immediately result in an unrelated instance of the current session and would thus force the simulator to redo the simulation work on this session.

So where in the schedule should the randomness of session (ℓ, i) be determined? On the one hand, we would like to determine the randomness of a session as late as possible (in order to maximize the effect of changes in the history of the interaction on the randomness of the session). On the other hand, we cannot afford to determine the randomness after the session's initiating message is scheduled (since the protocol's specification may require that the verifier's randomness be completely determined before the first verifier message is sent). For technical reasons, the point at which we choose to determine the randomness of session (ℓ, i) is the point at which recursive block number ℓ is invoked. That is, to obtain the randomness of session $(\ell, i) = \pi_{\mathrm{sn}}(\overline{q})$, we feed the hash function with the prefix of query $\overline{q}$ that ends just before the first message in block number ℓ. (This prefix is called the *block-prefix* of query $\overline{q}$ and is defined below.) In order to achieve independence with other sessions in block number ℓ, we will also feed the hash function with the value of i. This (together with the above choice) guarantees us the following properties: (1) The input to the

hash function (and thus the randomness for session (ℓ, i)) does not change once the interaction in the session begins (that is, once the first verifier message is sent). (2) For every pair of different sessions, the input to the hash function is different (and thus the randomness for each session is independent). (3) Even a single modification in the prefix of the interaction up to the first message in block number ℓ induces fresh randomness for all sessions in block number ℓ.

DEFINITION 4.2 (block-prefix). *The* block-prefix *of a query* $\overline{q}$ *satisfying* $\pi_{\mathrm{sn}}(\overline{q}) = (\ell, i)$ *is the prefix of* $\overline{q}$ *that is answered with the first verifier message of session* $(\ell, 1)$ *(that is, the first main session in block number* ℓ*). More formally,* $bp(\overline{q}) = (b_1, a_1, \ldots, b_\gamma, a_\gamma)$ *is the block-prefix of* $\overline{q} = (b_1, a_1, \ldots, b_t, a_t)$ *if* $\pi_{\mathrm{sn}}(bp(\overline{q})) = (\ell, 1)$ *and* $\pi_{\mathrm{msg}}(bp(\overline{q})) = 1$*. The block-prefix will be said to correspond to recursive block number* ℓ*.*[11] (Note that i may be any index in $\{1, \ldots, n\}$ and that a_t need not belong to session (ℓ, i).)

Determining whether and when to abort sessions. Whereas the randomness that is used to compute the honest verifier's answers in each session is determined before a session begins, the randomness that is used in order to decide whether to abort a session is chosen independently every time the execution of the schedule reaches the next verifier message in this session. As before, the required randomness is obtained by applying a hash function on the suitable prefix of the execution transcript. This time, however, the length of the prefix increases each time the execution of the session reaches the next verifier message (rather than being fixed for the whole execution of the session). This way, the decision of whether to abort a session also depends on the contents of messages that were exchanged after the initiation of the session has occurred. Specifically, in order to decide whether to abort session $(\ell, i) = \pi_{\mathrm{sn}}(\overline{q})$ at the jth message (where $j = \pi_{\mathrm{msg}}(\overline{q})$), we feed the hash function with the prefix (of query $\overline{q}$) that ends with the $(j-1)$st prover message in the nth main session of block number ℓ. (As before, the hash function is also fed with the value of i in order to achieve independence from other sessions in the block.) This prefix is called the *iteration-prefix* of query $\overline{q}$ and is defined next. (See Figure 4.2 for a graphical description of the block-prefix and iteration-prefix of a query.)

DEFINITION 4.3 (iteration-prefix). *The* iteration-prefix *of a query* $\overline{q}$ *satisfying* $\pi_{\mathrm{sn}}(\overline{q}) = (\ell, i)$ *and* $\pi_{\mathrm{msg}}(\overline{q}) = j > 1$ *is the prefix of* $\overline{q}$ *that ends with the* $(j-1)$*st prover message in session* (ℓ, n) *(that is, the* n*th main session in block number* ℓ*). More formally,* $ip(\overline{q}) = (b_1, a_1, \ldots, b_\delta, a_\delta)$ *is the iteration-prefix of* $\overline{q} = (b_1, a_1, \ldots, b_t, a_t)$ *if* a_δ *is of the form* $\mathtt{p}_{j-1}^{(n)}$ *(where* $\mathtt{p}_{j-1}^{(n)}$ *denotes the* $(j-1)$*st prover message in the* n*th main session of block number* ℓ*). This iteration-prefix is said to correspond to the block-prefix of* $\overline{q}$*.* (Again, note that i may be any index in $\{1, \ldots, n\}$ and that a_t need not belong to session (ℓ, i). Also note that the iteration-prefix is defined only for $\pi_{\mathrm{msg}}(\overline{q}) > 1$.)

We stress that two queries $\overline{q}_1, \overline{q}_2$ may have the same iteration-prefix even if they do not correspond to the same session. This could happen whenever $bp(\overline{q}_1) = bp(\overline{q}_2)$ and $\pi_{\mathrm{msg}}(\overline{q}_1) = \pi_{\mathrm{msg}}(\overline{q}_2)$ (which is possible even if $\pi_{\mathrm{sn}}(\overline{q}_1) \neq \pi_{\mathrm{sn}}(\overline{q}_2)$).

Motivating Definitions 4.2 *and* 4.3. The choices made in Definitions 4.2 and 4.3 are designed to capture the difficulties encountered whenever many sessions are to be simulated concurrently. As was previously mentioned, we would like to create a situation in which every attempt of the simulator to rewind a specific session will result in a loss of work done for other sessions (and so will cause the simulator to do

[11] In the special case that $\ell = 1$ (that is, we are in the first block of the schedule), we define $bp(\overline{q}) = \perp$.

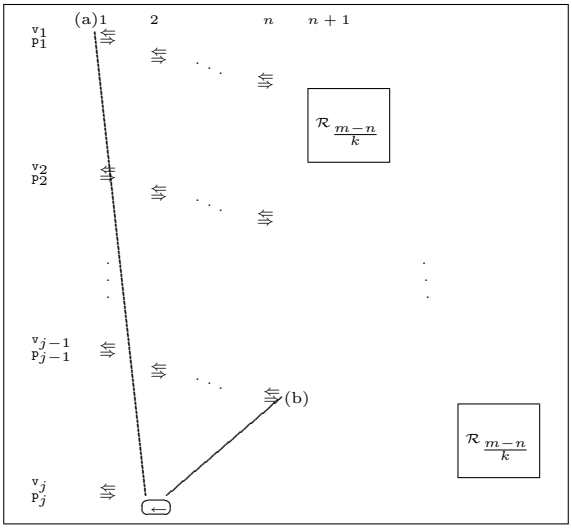


FIG. 4.2. *Determining the prefixes of query $\overline{q}$ (in this example, query $\overline{q}$ ends with a $\mathtt{p}_j^{(1)}$ message and is to be answered by $\mathtt{v}_j^{(2)}$, represented by the marked arrow):* (a) *indicates the* block-prefix *of $\overline{q}$ (i.e., messages up to this point are used by $V_{g,h}$ to determine the randomness to be used for computing message $\mathtt{v}_j^{(2)}$).* (b) *indicates the* iteration-prefix *of $\overline{q}$ (i.e., messages up to this point are used by $V_{g,h}$ to determine whether or not message $\mathtt{v}_j^{(2)}$ will be set to* `ABORT`*).*

the same amount of work all over again). In order to force the simulator to repeat each such rewinding attempt many times, we make each rewinding attempt fail with some predetermined probability (by letting the verifier send an `ABORT` message instead of a legal answer).[12]

To see that Definitions 4.2 and 4.3 indeed lead to the fulfillment of the above requirements, we consider the following example. Suppose that, at some point during the simulation, the adversary verifier aborts session (ℓ, i) at the jth message (while answering query $\overline{q}$). Further suppose that (for some unspecified reason) the simulator wants to get a "second chance" in receiving a legal answer to the jth message in session (ℓ, i) (hoping that it will not receive the `ABORT` message again). Recall that the decision of whether to abort a session depends on the outcome of a hash function when applied to the iteration-prefix $ip(\overline{q})$ of query $\overline{q}$. In particular, to obtain a "second chance," the black-box simulator has no choice but to change at least one prover message in the above iteration-prefix. (In other words, the simulator must rewind the interaction to some message occurring in iteration-prefix $ip(\overline{q})$.) At first glance, it may seem that the effect of changes in the iteration-prefix of query $\overline{q}$ is confined to the messages that belong to session $(\ell, i) = \pi_{\mathrm{sn}}(\overline{q})$ (or at most to messages that belong to other sessions in block number ℓ). However, taking a closer look at the schedule, we observe that every iteration-prefix (and, in particular, $ip(\overline{q})$) can also be viewed as the block-prefix of a recursive block one level down in the recursive construction. Viewed this way, it is clear that the effect of changes in $ip(\overline{q})$ is not confined only to messages that correspond to recursive block number ℓ but rather extends also to sessions at lower levels in the recursive schedule. By changing even a single message

[12]Recall that all of the above is required in order to make the simulator's work accumulate too much and eventually cause its running time to be superpolynomial.

in iteration-prefix $ip(\overline{q})$, the simulator is actually modifying the block-prefix of all recursive blocks in a subschedule one level down in the recursive construction. This means that the randomness for all sessions in these blocks is completely modified (recall that the randomness of a session is determined by applying a hash function on the corresponding block-prefix) and that all the simulation work done for these sessions is lost. In particular, by changing even a single message in iteration-prefix $ip(\overline{q})$, the simulator will find himself doing the simulation work for these lower-level sessions all over again.

Having established the effect of changes in iteration-prefix $ip(\overline{q})$ on sessions at lower levels in the recursive schedule, we now turn to examine the actual effect on session $(\ell, i) = \pi_{\mathrm{sn}}(\overline{q})$ itself. One possible consequence of changes in iteration-prefix $ip(\overline{q})$ is that they may also affect the contents of the block-prefix $bp(\overline{q})$ of query $\overline{q}$. (Notice that, by definition, the block-prefix $bp(\overline{q})$ of query $\overline{q}$ is contained in the iteration-prefix $ip(\overline{q})$ of query $\overline{q}$.) Whenever this happens, the randomness used for session (ℓ, i) is completely modified, and all simulation work done for this session will be lost. A more interesting consequence of a change in the contents of iteration-prefix $ip(\overline{q})$ is that it will result in a completely independent decision of whether session (ℓ, i) is to be aborted at the jth message. (The decision of whether to abort is taken whenever the simulator makes a query $\overline{q}$ satisfying $\pi_{\mathrm{sn}}(\overline{q}) = (\ell, i)$ and $\pi_{\mathrm{msg}}(\overline{q}) = j$.) In other words, each time the simulator attempts to get a "second chance" in receiving a legal answer to the jth message in session (ℓ, i) (by rewinding the interaction to a message that belongs to iteration-prefix $ip(\overline{q})$), it faces the risk of being answered with an `ABORT` message independently of all previous rewinding attempts.

4.1.3. The actual verifier strategy $V_{g,h}$. We consider what happens when a simulator S (for the above schedule) is given oracle access to a verifier strategy $V_{g,h}$ defined as follows (depending on hash functions g, h and the input x). Recall that we may assume that S runs in strict polynomial time: we denote such time bound by $t_S(\cdot)$. Let G denote a small family of $t_S(n)$-wise independent hash functions mapping $\mathrm{poly}(n)$-bit long sequences into a single bit of output so that, for every α, we have $\Pr_{g \leftarrow G}[g(\alpha) = 1] = n^{-1/2k}$. Let H denote a small family of $t_S(n)$-wise independent hash functions mapping $\mathrm{poly}(n)$-bit long sequences to $\rho_V(n)$-bit sequences so that, for every α, we have $\Pr_{h \leftarrow H}[h(\alpha) = 1] = 2^{-\rho_V(n)}$ (where $\rho_V(n)$ is the number of random bits used by an honest verifier V on an input $x \in \{0,1\}^n$).[13] We describe a family $\{V_{g,h}\}_{g \in G, h \in H}$ of adversarial verifier strategies (where x is implicit in $V_{g,h}$). On query $\overline{q} = (b_1, a_1, \ldots, a_{t-1}, b_t, a_t)$, the verifier acts as follows:

1. First, $V_{g,h}$ checks whether the execution transcript given by the query is legal (i.e., corresponds to a possible execution prefix) and halts with a special `ERROR` message if the query is not legal.[14]
2. Next, $V_{g,h}$ determines the block-prefix, $bp(\overline{q}) = (b_1, a_1, \ldots, b_\gamma, a_\gamma)$, of query $\overline{q}$. It also determines the identifiers of the next message $(\ell, i) = \pi_{\mathrm{sn}}(\overline{q})$ and $j = \pi_{\mathrm{msg}}(\overline{q})$, the iteration-prefix $ip(\overline{q}) = (b_1, a_1, \ldots, b_\delta, \mathtt{p}_{j-1}^{(n)})$, and the $j-1$ prover messages of session i appearing in query $\overline{q}$ (which we denote by $\mathtt{p}_1^{(i)}, \ldots, \mathtt{p}_{j-1}^{(i)}$).

[13] We stress that functions in such families can be described by strings of polynomial length in a way that enables polynomial-time evaluation (cf. [24, 9, 10, 1]).

[14] In particular, $V_{g,h}$ checks whether the query is of the prescribed format (as described in section 2.5 and as determined by the schedule) and whether the content of its messages is consistent with $V_{g,h}$'s prior answers. (That is, for every proper prefix $\overline{q}' = (b_1, a_1, \ldots, b_u, a_u)$ of query $\overline{q} = (b_1, a_1, \ldots, b_t, a_t)$, the verifier checks whether the value of b_{u+1} (as it appears in $\overline{q}$) is indeed equal to the value of $V_{g,h}(\overline{q}')$.)

(*Motivating discussion.* The next message is the jth verifier message in the ith session of block ℓ. The value of the block-prefix, $bp(\overline{q})$, is used in order to determine the randomness of session (ℓ, i), whereas the value of the iteration-prefix, $ip(\overline{q})$, is used in order to determine whether session (ℓ, i) is about to be aborted at this point (i.e., jth message) in the schedule (by answering with a special `ABORT` message).)

3. If $j = 1$, then $V_{g,h}$ answers with the verifier's fixed initiation message for session i (i.e., $\mathtt{v}_1^{(i)}$).
4. If $j > 1$, then $V_{g,h}$ determines $b_{i,j} = g(i, ip(\overline{q}))$ (i.e., a bit deciding whether to abort session i):
   (a) If $b_{i,j} = 0$, then $V_{g,h}$ sets $\mathtt{v}_j^{(i)} = \mathtt{ABORT}$ (indicating that $V_{g,h}$ aborts session i).
   (b) If $b_{i,j} = 1$, then $V_{g,h}$ determines $r_i = h(i, bp(\overline{q}))$ (as coins to be used by V) and computes the message $\mathtt{v}_j^{(i)} = V(x, r_i; \mathtt{p}_1^{(i)}, \ldots, \mathtt{p}_{j-1}^{(i)})$ that would have been sent by the honest verifier on common input x, random-pad r_i, and prover's messages $\mathtt{p}_1^{(i)}, \ldots, \mathtt{p}_{j-1}^{(i)}$.
   (c) Finally, $V_{g,h}$ answers with $\mathtt{v}_j^{(i)}$.

Dealing with `ABORT` *messages.* Note that, once $V_{g,h}$ has aborted a session, the interaction in this session essentially stops, and there is no need to continue exchanging messages in this session. However, for simplicity of exposition, we assume that the verifier and prover stick to the fixed schedule of section 4.1.1 and exchange `ABORT` messages whenever an aborted session is scheduled. Specifically, if the jth verifier message in session i is `ABORT`, then all subsequent prover and verifier messages in that session will also equal `ABORT`.

On the arguments to g and h. The hash function h, which determines the random input for V in a session, is applied both on i (the identifier of the relevant session within the current block) and on the entire block-prefix of the query $\overline{q}$. This means that, even though all sessions in a specific block have the same block-prefix, for every pair of two different sessions, the corresponding random inputs of V will be independent of each other (as long as the number of applications of h does not exceed $t_S(n)$, which is indeed the case in our application). The hash function g, which determines whether and when the verifier aborts sessions, is applied both on i and on the entire iteration-prefix of the query $\overline{q}$. As in the case of h, the decision whether to abort a session is independent from the same decision for other sessions (again, as long as g is not applied more than $t_S(n)$ times). However, there is a significant difference between the inputs of h and g: Whereas the input of h is *fixed* once i and the block-prefix are fixed (and is unaffected by mesages that belong to that session), the input of g *varies* depending on previous messages sent in that session. In particular, whereas the randomness of a session is completely determined once the session begins, the decision of whether to abort a session is taken independently each time that the schedule reaches the next verifier message of this session.

On the number of different prefixes that occur in interactions with $V_{g,h}$. Since the number of recursive blocks in the schedule is equal to n, and since there is a one-to-one correspondence between recursive blocks and block-prefixes, we have that the number of different block-prefixes that occur during an interaction between an *honest prover* P and the verifier $V_{g,h}$ is always equal to n. Since the number of iterations in the message exchange phase of a recursive invocation of the schedule equals $k + 1$, and since there is a one-to-one correspondence between such iterations and iteration-

prefixes,[15] we have that the number of different iteration-prefixes that occur during an interaction between an honest prover P and the verifier $V_{g,h}$ is always equal to $k \cdot n$ (that is, k different iteration-prefixes for each one of the n recursive invocations of the schedule). In contrast, the number of different block-prefixes (resp., iteration-prefixes) that occur during an execution of a black-box simulator S that is given oracle access to $V_{g,h}$ may be considerably larger than n (resp., $k \cdot n$). The reason for this is that there is nothing that prevents the simulator from feeding $V_{g,h}$ with different queries of the same length. (This corresponds to the so-called rewinding of an interaction.) Still, the number of different prefixes in an execution of S is always upper bounded by the running time of S; that is, $t_S(n)$.

On the probability that a session is never aborted. A typical interaction between an honest prover P and the verifier $V_{g,h}$ will contain sessions whose execution has been aborted prior to completion. Recall that, at each point in the schedule, the decision of whether or not to abort the next scheduled session depends on the outcome of g. Since the function g returns 1 with probability $n^{-1/2k}$, a specific session is never aborted with probability $(n^{-1/2k})^k = n^{-1/2}$. Using the fact that, whenever a session is not aborted, $V_{g,h}$ operates as the honest verifier, we infer that the probability that a specific session is eventually accepted by $V_{g,h}$ is at least $1/2$ times the probability that the very same session is never aborted (where $1/2$ is an arbitrary lower bound on the completeness probability of the protocol). In other words, the probability that a session is accepted by $V_{g,h}$ is at least $\frac{n^{-1/2}}{2}$. In particular, for every set of n sessions, the expected number of sessions that are eventually accepted by $V_{g,h}$ (when interacting with the honest prover P) is at least $n \cdot \frac{n^{-1/2}}{2} = \frac{n^{1/2}}{2}$, and, with overwhelming high probability, at least $\frac{n^{1/2}}{4}$ sessions are accepted by $V_{g,h}$.

A slight modification of the verifier strategy. To facilitate the analysis, we slightly modify the verifier strategy $V_{g,h}$ so that it does not allow the number of accepted sessions in the history of the interaction to deviate much from its "expected behavior." Loosely speaking, given a prefix of the execution transcript (ending with a prover message), the verifier will check whether the recursive block that has just been completed contains at least $\frac{n^{1/2}}{4}$ accepted sessions. (To this end, it will be sufficient to inspect the history of the interaction only when the execution of the schedule reaches the end of a recursive block—that is, whenever the schedule reaches the last prover message in the last session of a recursive block (i.e., some $\mathtt{p}_{k+1}^{(n)}$ message).) The modified verifier strategy (which we continue to denote by $V_{g,h}$) is obtained by adding to the original strategy an additional step 1' (to be executed after step 1 of $V_{g,h}$):

1'. If a_t is of the form $\mathtt{p}_{k+1}^{(n)}$ (i.e., in case query $\overline{q} = (b_1, a_1, \ldots, b_t, a_t)$ ends with the last prover message of the nth main session of a recursive block), $V_{g,h}$ checks whether the transcript $\overline{q} = (b_1, a_1, \ldots, b_t, \mathtt{p}_{k+1}^{(n)})$ contains the accepting conversations of at least $\frac{n^{1/2}}{4}$ main sessions in the block that has just been completed. In case it does not, $V_{g,h}$ halts with a special `DEVIATION` message (indicating that the number of accepted sessions in the block that has just been completed deviates from its expected value).

Motivating discussion. Since the expected number of accepted sessions in a specific block is at least $\frac{n^{1/2}}{2}$, the probability that the block contains less than $\frac{n^{1/2}}{4}$

[15] The only exception is the first iteration in the message exchange phase. Since only queries $\overline{q}$ that satisfy $\pi_{\mathrm{msg}}(\overline{q}) > 1$ have an iteration-prefix, the first iteration will never have a corresponding iteration-prefix.

accepted sessions is negligible. Still, the above modification is not superfluous (even though it refers to events that occur only with negligible probability); it allows us to assume that every recursive block that is completed *during the simulation* (including those that *do not* appear in the simulator's output) contains at least $\frac{n^{1/2}}{4}$ accepted sessions. In particular, whenever the simulator feeds $V_{g,h}$ with a partial execution transcript (i.e., a query), we are guaranteed that, for every completed block in this transcript, the simulator has indeed "invested work" to simulate the at least $\frac{n^{1/2}}{4}$ accepted sessions in the block.

A slight modification of the simulator. Before presenting the decision procedure, we slightly modify the simulator so that it never makes a query that is answered with either the `ERROR` or the `DEVIATION` messages by the verifier $V_{g,h}$. Note that the corresponding condition can be easily checked by the simulator (which can easily produce this special message by itself)[16] and that the modification does not affect the simulator's output. From this point on, when we talk of the simulator (which we continue to denote by S), we mean the modified one.

4.2. The decision procedure for *L*. We are now ready to describe a probabilistic polynomial-time decision procedure for L, based on the black-box simulator S and the verifier strategies $V_{g,h}$. On input $x \in \{0,1\}^n$, the procedure operates as follows:

1. Uniformly select hash functions $g \stackrel{R}{\leftarrow} G$ and $h \stackrel{R}{\leftarrow} H$.
2. Invoke S on input x, providing it with black-box access to $V_{g,h}$ (as defined above). That is, the procedure emulates the execution of the oracle machine S on input x along with emulating the answers of $V_{g,h}$, where g and h are as determined in step 1.
3. Accept iff S outputs a legal transcript (as determined by steps 1 and 1' of $V_{g,h}$).[17]

By our hypothesis, the above procedure runs in probabilistic polynomial time. We next analyze its performance.

LEMMA 4.4 (performance on YES-instances). *For all but finitely many $x \in L$, the above procedure accepts x with probability at least* 2/3.

Proof sketch. Let $x \in L$, $g \stackrel{R}{\leftarrow} G$, and $h \stackrel{R}{\leftarrow} H$, and consider the honest prover P. We show below that, except for negligible probability (where the probability is taken over the random choices of g, h, and P's coin tosses), when $V_{g,h}$ interacts with P, all recursive blocks in the resulting transcript contain the accepting conversations of at least $\frac{n^{1/2}}{4}$ main sessions. Since, for *every* g and h, the simulator $S^{V_{g,h}}(x)$ must generate a transcript whose deviation gap from $\langle P, V_{g,h}\rangle(x)$ is at most 1/4, it follows that $S^{V_{g,h}}(x)$ has deviation gap at most 1/4 from $\langle P, V_{g,h}\rangle(x)$ when $g \stackrel{R}{\leftarrow} G$ and $h \stackrel{R}{\leftarrow} H$ also. Consequently, when S is run by the decision procedure for L, the transcript $S^{V_{g,h}}(x)$ will not be legal with probability at most 1/3. Details follow.

[16] We stress that, as opposed to the `ERROR` and `DEVIATION` messages, the simulator cannot predict whether its query is about to be answered with the `ABORT` message.

[17] Recall that we are assuming that the simulator never makes a query that is ruled out by steps 1 and 1' of $V_{g,h}$. Since, before producing output $(b_1, a_1, \ldots, b_T, a_T)$, the simulator makes the query $(b_1, a_1, \ldots, b_T, a_T)$, checking the legality of the transcript in step 3 is not really necessary (as, in case that the modified simulator indeed reaches the output stage "safely," we are guaranteed that it will produce a legal output). In particular, we are always guaranteed that the simulator either produces execution transcripts in which every recursive block contains at least $n^{1/2}/4$ sessions that were accepted by $V_{g,h}$ or does not produce any output at all.

Let τ denote the random variable describing the transcript of the interaction between the honest prover P and $V_{g,h}$, where the probability is taken over the choices of g, h, and P. Let $s \in \{1, \ldots, n^2\}$. We first calculate the probability that the sth session in τ is completed and accepted (i.e., $V_{g,h}$ sends the message $\mathtt{v}_{k+1}^{(s)} = \texttt{ACCEPT}$), conditioned on the event that $V_{g,h}$ did not abandon the interaction beforehand (i.e., $V_{g,h}$ did not send the `DEVIATION` message before).[18] For uniformly selected $g \stackrel{R}{\leftarrow} G$, the probability that $V_{g,h}$ does not abort the session in each of the k rounds, given that it has not already aborted, is $n^{-1/2k}$. Thus, conditioned on the event that $V_{g,h}$ did not output `DEVIATION` beforehand, the session is completed (without being aborted) with probability $(n^{-1/2k})^k = n^{-1/2}$.

The key observation is that, if h is uniformly chosen from H, then, conditioned on the event that $V_{g,h}$ did not output `DEVIATION` beforehand and the current session is not aborted, the conversation between $V_{g,h}$ and P is distributed identically to the conversation between the honest verifier V and P on input x. By the completeness requirement for zero-knowledge protocols, we have that V accepts in such an interaction with probability at least $1/2$. (This probability is actually higher, but $1/2$ is more than enough for our purposes.) Consequently, for uniformly selected g and h, conditioned on the event that $V_{g,h}$ did not output `DEVIATION` beforehand, the probability that a session is accepted by $V_{g,h}$ is at least $\frac{n^{-1/2}}{2}$.

We calculate the probability that τ contains a block such that less than $\frac{n^{1/2}}{4}$ of its sessions are accepted. Say that a block B in a transcript has been completed if all the messages of sessions in B have been sent during the interaction. Say that B is admissible if the number of accepted sessions that belong to block B in the transcript is at least $\frac{n^{1/2}}{4}$. Enumerating blocks in the order in which they are completed (that is, when we refer to the ℓth block in τ, we mean the ℓth block that is completed in τ), we denote by γ_ℓ the event that all of the blocks up to and including the ℓth block are admissible in τ.

For $i \in \{1, \ldots, n\}$, define a boolean indicator α_i^ℓ to be 1 iff the ith session in the ℓth block is accepted by $V_{g,h}$. We have seen that, conditioned on the event $\gamma_{\ell-1}$, each α_i^ℓ is 1 with probability at least $\frac{n^{-1/2}}{2}$. As a consequence, for every ℓ, the expectation of $\sum_{i=1}^n \alpha_i^\ell$ (i.e., the number of accepted main sessions in block number ℓ) is at least $\frac{n^{1/2}}{2}$. Since, conditioned on $\gamma_{\ell-1}$, the α_i^ℓ's are independent of each other, we can apply the Chernoff bound and infer that $\Pr[\gamma_\ell | \gamma_{\ell-1}] > 1 - e^{-\Omega(n^{1/2})}$. Furthermore, since no session belongs to more than one block, we have $\Pr[\gamma_\ell] \geq \Pr[\gamma_l | \gamma_{\ell-1}] \cdot \Pr[\gamma_{l-1}]$. It follows (by induction on the number of completed blocks in a transcript) that all blocks in τ are admissible with probability at least $(1 - e^{-\Omega(n^{1/2})})^n > 1 - n \cdot e^{-\Omega(n^{1/2})}$. The lemma follows. □

LEMMA 4.5 (performance on NO-instances). *For all but finitely many $x \notin L$, the above procedure rejects x with probability at least* $2/3$.

We can actually prove that, for every positive polynomial $p(\cdot)$ and for all but finitely many $x \notin L$, the above procedure accepts x with probability at most $1/p(|x|)$. Assuming toward contradiction that this is not the case, we will construct a (probabilistic polynomial-time) strategy for a cheating prover that fools the honest verifier V with success probability at least $1/\mathrm{poly}(n)$ in contradiction to the soundness (and

[18] Note that, since we are dealing with the honest prover P, there is no need to consider the `ERROR` message at all (since, in an interaction with the honest prover P, the adversary verifier $V_{g,h}$ will never output `ERROR` anyway).

even computational soundness) of the proof-system.

5. Proof of Lemma 4.5 (performance on NO-instances). Let us fix an $x \in \{0,1\}^n \setminus L$ as above.[19] Denote by $\mathtt{AC} = \mathtt{AC}_x$ the set of triplets (σ, g, h) so that, on input x, internal coins σ, and oracle access to $V_{g,h}$, the simulator outputs a legal transcript (which we denote by $S_\sigma^{V_{g,h}}(x)$). Recall that our contradiction assumption is that $\Pr_{\sigma,g,h}[(\sigma,g,h) \in \mathtt{AC}] > 1/p(n)$ for some fixed positive polynomial $p(\cdot)$. Before proceeding with the proof of Lemma 4.5, we formalize what we mean by referring to the "execution of the simulator."

DEFINITION 5.1 (execution of the simulator). *Let $x, \sigma \in \{0,1\}^*$, $g \in G$, and $h \in H$. The* execution *of simulator S, denoted $\mathrm{EXEC}_x(\sigma, g, h)$, is the sequence of queries made by S, given input x, random coins σ, and oracle access to $V_{g,h}(x)$.*

Since the simulator has the ability to "rewind" the verifier $V_{g,h}$ and explore $V_{g,h}$'s output on various execution prefixes (i.e., queries) of the same length, the number of distinct block-prefixes that appear in $\mathrm{EXEC}_x(\sigma, g, h)$ may be strictly larger than n. (Recall that the schedule consists of n invocations to recursive blocks and that, in an interaction between the honest prover P and $V_{g,h}$, there is a one-to-one correspondence between recursive blocks and block-prefixes.) As a consequence, the ℓth distinct block-prefix appearing in $\mathrm{EXEC}_x(\sigma, g, h)$ does not necessarily correspond to the ℓth recursive block in the schedule. Nevertheless, given $\mathrm{EXEC}_x(\sigma, g, h)$ and ℓ, one can easily determine for the ℓth distinct block-prefix in the execution of the simulator the index of its corresponding block in the schedule (say, by extracting the ℓth distinct block-prefix in $\mathrm{EXEC}_x(\sigma, g, h)$ and then analyzing its length).

In what follows, given a specific block-prefix $\overline{bp}$, we let $\ell^{(\overline{bp})} \in \{1, \ldots, n\}$ denote the index of its corresponding block in the schedule (as determined by $\overline{bp}$'s length). Note that two different block-prefixes $\overline{bp}_1$ and $\overline{bp}_2$ in $\mathrm{EXEC}_x(\sigma, g, h)$ may satisfy $\ell^{(\overline{bp}_1)} = \ell^{(\overline{bp}_2)}$ (as they may correspond to two different instances of the same recursive block). In particular, session $(\ell^{(\overline{bp}_1)}, i)$ may have more than a single occurrence during the execution of the simulator (whereas, in an interaction of the honest prover P with $V_{g,h}$, each session index will occur exactly once). This means that, whenever we refer to an instance of session (ℓ, i) in the simulation, we will also have to explicitly specify to which block-prefix this instance corresponds.

In order to avoid cumbersome statements, we will abuse the notation $\ell^{(\overline{bp})}$ and also use it in order to specify to which instance the recursive block $\ell^{(\overline{bp})}$ corresponds. That is, whenever we refer to recursive block number $\ell^{(\overline{bp})}$, we will actually mean "the specific instance of recursive block number ℓ $(= \ell^{(\overline{bp})})$ that corresponds to block-prefix $\overline{bp}$ in $\mathrm{EXEC}_x(\sigma, g, h)$." Viewed this way, for $\ell^{(\overline{bp}_1)} = \ell^{(\overline{bp}_2)}$, sessions $(\ell^{(\overline{bp}_1)}, i)$ and $(\ell^{(\overline{bp}_2)}, i)$ actually correspond to two different instances of the same session in the schedule.

5.1. The cheating prover. The cheating prover (denoted by P^*) starts by uniformly selecting a triplet (σ, g, h) while hoping that $(\sigma, g, h) \in \mathtt{AC}$. It next uniformly selects a pair $(\xi, \eta) \in \{1, \ldots, t_S(n)\} \times \{1, \ldots, n\}$, where the simulator's running time, $t_S(n)$, acts as a bound on the number of (different block-prefixes induced by the)

[19] Actually, we need to consider infinitely many such x's.

queries made by S on input $x \in \{0,1\}^n$. The prover next emulates an execution of $S_\sigma^{V_{g,h^{(r)}}}(x)$ (where $h^{(r)}$, which is essentially equivalent to h, will be defined below) while interacting with $V(x,r)$ (that is, the honest verifier, running on input x and using coins r). The prover handles the simulator's queries as well as the communication with the verifier as follows: Suppose that the simulator makes query $\overline{q} = (b_1, a_1, \ldots, b_t, a_t)$, where the a's are prover messages.

1. Operating as $V_{g,h}$, the cheating prover determines the block-prefix $bp(\overline{q}) = (b_1, a_1, \ldots, b_\gamma, a_\gamma)$. It also determines $(\ell, i) = \pi_{\mathrm{sn}}(\overline{q})$, $j = \pi_{\mathrm{msg}}(\overline{q})$, the iteration-prefix $ip(\overline{q}) = (b_1, a_1, \ldots, b_\delta, \mathtt{p}_{j-1}^{(n)})$, and the $j-1$ prover messages $\mathtt{p}_1^{(i)}, \ldots, \mathtt{p}_{j-1}^{(i)}$ appearing in the query $\overline{q}$ (as done by $V_{g,h}$ in step 2). (Note that, by the modification of S, there is no need to perform steps 1 and 1' of $V_{g,h}$.)
2. If $j = 1$, the cheating prover answers the simulator with the verifier's fixed initiation message for session i (as done by $V_{g,h}$ in step 3).
3. If $j > 1$, the cheating prover determines $b_{i,j} = g(i, ip(\overline{q}))$ (as done by $V_{g,h}$ in step 4).
4. If $bp(\overline{q})$ is the ξth distinct block-prefix resulting from the simulator's queries so far and if, in addition, i equals η, then the cheating prover operates as follows:
   (a) If $b_{i,j} = 0$, then the cheating prover answers the simulator with `ABORT`.
   Motivating discussion for substeps b and c. The cheating prover has now reached a point in the schedule at which it is supposed to feed the simulator with $\mathtt{v}_j^{(i)}$. To do so, it first forwards $\mathtt{p}_{j-1}^{(i)}$ to the honest verifier $V(x,r)$ and only then feeds the simulator with the verifier's answer $\mathtt{v}_j^{(i)}$ (as if it were the answer given by $V_{g,h^{(r)}}$). We stress the following two points: (1) The cheating prover cannot forward more than one $\mathtt{p}_{j-1}^{(i)}$ message to V (since P^* and V engage in an actual execution of the protocol $\langle P, V\rangle$). (2) The cheating prover will wait and forward $\mathtt{p}_{j-1}^{(i)}$ to the verifier only when $\mathtt{v}_j^{(i)}$ is the next scheduled message.
   (b) If $b_{i,j} = 1$ and the cheating prover has sent only $j-2$ messages to the actual verifier, the cheating prover forwards $\mathtt{p}_{j-1}^{(i)}$ to the verifier and feeds the simulator with the verifier's response (i.e., which is of the form $\mathtt{v}_j^{(i)}$).[20]
   (We comment that, by our conventions regarding the simulator, it cannot be the case that the cheating prover has sent less than $j-2$ prover messages to the actual verifier. The prefixes of the current query dictate $j-2$ sequences of prover messages with distinct lengths so that none of these sequences was answered with `ABORT`. In particular, the last message of each one of these sequences was already forwarded to the verifier.)
   (c) If $b_{i,j} = 1$ and the cheating prover has already sent $j-1$ messages (or more) to the actual verifier, then it retrieves the $(j-1)$st answer it has received and feeds it to the simulator.

[20] Note that, in the special case that $j = 1$ (i.e., when the verifier's response is the fixed initiation message $\mathtt{v}_1^{(i)}$), the cheating prover cannot really forward $\mathtt{p}_{j-1}^{(i)}$ to the honest verifier (since no such message exists). Still, since $\mathtt{v}_1^{(i)}$ is a fixed initiation message, the cheating prover can produce $\mathtt{v}_1^{(i)}$ without actually having to interact with the honest verifier (as it indeed does in step 2 of the cheating prover strategy).

(We comment that this makes sense provided that the simulator never makes two queries with the same block-prefix and the same number of prover messages but with a different sequence of such messages. However, for $j \geq 2$, it may be the case that a previous query regarding the same block-prefix had a different $\mathtt{p}_{j-1}^{(i)}$ message. This is the case in which the cheating prover may fail to conduct step 4c (see further discussion below).)

5. If either $bp(\overline{q})$ is NOT the ξth distinct block-prefix resulting from the queries so far, or if i is NOT equal to η, the prover emulates $V_{g,h}$ in the obvious manner (i.e., as in step 4 of $V_{g,h}$):
   (a) If $b_{i,j} = 0$, then the cheating prover answers the simulator with `ABORT`.
   (b) If $b_{i,j} = 1$, then the cheating prover determines $r_i = h(i, bp(\overline{q}))$ and then answers the simulator with $V(x, r_i; \mathtt{p}_1^{(i)}, \ldots, \mathtt{p}_{j-1}^{(i)})$, where all notation is as above.

On the efficiency of the cheating prover. Notice that the strategy of the cheating prover can be implemented in polynomial time (that is, given that the simulator's running time, $t_S(\cdot)$, is polynomial as well). Thus Lemma 4.5 (and so Theorem 1.1) will also hold if $\langle P, V \rangle$ is an *argument* system (since, in the case of argument systems, the existence of an *efficient* P^* leads to a contradiction of the computational soundness of $\langle P, V \rangle$).

The cheating prover may "do nonsense" in step 4c. The cheating prover is hoping to convince an honest verifier by focusing on the ηth session in recursive block number $\ell^{(\overline{bp}_\xi)}$, where $\overline{bp}_\xi$ denotes the ξth distinct block-prefix in the simulator's execution. Prover messages in session $(\ell^{(\overline{bp}_\xi)}, \eta)$ are received from the (multisession) simulator and are forwarded to the (single-session) verifier. The honest verifier's answers are then fed back to the simulator as if they were answers given by $V_{g,h^{(r)}}$ (defined below). For the cheating prover to succeed in convincing the honest verifier, the following two conditions must be satisfied: (1) Session $(\ell^{(\overline{bp}_\xi)}, \eta)$ is eventually accepted by $V_{g,h^{(r)}}$. (2) The cheating prover never "does nonsense" in step 4c during its execution. Let us clarify the meaning of this "nonsense."

One main problem that the cheating prover is facing while conducting step 4c emerges from the following fact: Whereas the black-box simulator is allowed to "rewind" $V_{g,h^{(r)}}$ (impersonated by the cheating prover) and attempt different execution prefixes before proceeding with the interaction of a session, the prover cannot do so while interacting with the actual verifier. In particular, the cheating prover may reach step 4c with a $\mathtt{p}_{j-1}^{(\eta)}$ message that is different from the $\mathtt{p}_{j-1}^{(\eta)}$ message that was previously forwarded to the honest verifier (in step 4b). Given that the verifier's answer to the current $\mathtt{p}_{j-1}^{(\eta)}$ message is most likely to be different from the answer which was given to the "previous" $\mathtt{p}_{j-1}^{(\eta)}$ message, by answering (in step 4c) in the same way as before, the prover action "makes no sense."[21]

We stress that, at this point in its execution, the cheating prover might as well have stopped with some predetermined "failure" message (rather than "doing non-

[21] We stress that the cheating prover does not know the random coins of the honest verifier, and so it cannot compute the verifier's answers by itself. In addition, since P^* and V are engaging in an actual execution of the specified protocol $\langle P, V \rangle$ (in which every message is sent exactly once), the cheating prover cannot forward the "recent" $\mathtt{p}_{j-1}^{(\eta)}$ message to the honest verifier in order to obtain the corresponding answer (because it has already forwarded the previous $\mathtt{p}_{j-1}^{(\eta)}$ message to the honest verifier).

sense"). However, for simplicity of presentation, it is more convenient for us to let the cheating prover "do nonsense."

The punchline of the analysis is that, with noticeable probability (over choices of (σ, g, h)), there exists a choice of (ξ, η) so that the above "bad" event will not occur for session $(\ell^{(\overline{bp}_\xi)}, \eta)$. That is, using the fact that the success of a "rewinding" also depends on the output of g (which determines whether and when sessions are aborted), we show that, with nonnegligible probability, step 4c is never reached with two different $\mathtt{p}_{j-1}^{(\eta)}$ messages. Specifically, for every $j \in \{2, \ldots, k+1\}$, once a $\mathtt{p}_{j-1}^{(\eta)}$ message is forwarded to the verifier (in step 4b), all subsequent $\mathtt{p}_{j-1}^{(\eta)}$ messages either are equal to the forwarded message or are answered with `ABORT`. (Here we assume that session $(\ell^{(\overline{bp}_\xi)}, \eta)$ is eventually accepted by $V_{g,h^{(r)}}$, and every $\mathtt{p}_{j-1}^{(\eta)}$ message is forwarded to the verifier at least once.)

Defining $h^{(r)}$ (mentioned above). Let (σ, g, h) and (ξ, η) be the initial choices made by the cheating prover, let $\overline{bp}_\xi$ be the ξth block-prefix appearing in $\text{EXEC}_x(\sigma, g, h)$, and suppose that the honest verifier uses coins r. Then the function $h^{(r)} = h^{(r,\sigma,g,h,\xi,\eta)}$ is defined to be uniformly distributed among the functions h', which satisfy the following conditions: The value of h', when applied on $(\eta, \overline{bp}_\xi)$, equals r, whereas, for $(\eta', \xi') \neq (\eta, \xi)$, the value of h', when applied on $(\eta', \overline{bp}_{\xi'})$, equals the value of h on this prefix. (The set of such functions h' is not empty due to the hypothesis that the functions are selected in a family of $t_S(n)$-wise independent hash functions.) We note that replacing h by $h^{(r)}$ does not affect step 5 of the cheating prover and that the cheating prover does not know $h^{(r)}$. In particular, whenever the honest verifier V uses coins r, one may think of the cheating prover as if it is answering the simulator's queries with the answers that would have been given by $V_{g,h^{(r)}}$.

CLAIM 5.2. *For every value of σ, g, ξ, and η, if h and r are uniformly distributed, then so is $h^{(r)}$.*

Proof sketch. Fix some simulator coins $\sigma \in \{0,1\}^*$ and $g \in G$, block-prefix index $\xi \in \{1, \ldots, t_S(n)\}$, and session index $\eta \in \{1, \ldots, n\}$. The key for proving Claim 5.2 is to view the process of picking a function $h \in H$ as consisting of two stages. The first stage is an iterative process in which up to $t_S(n)$ different arguments are adversarially chosen, and, for each such argument, the value of h on this argument is uniformly selected in its range. In the second stage, a function h is chosen uniformly from all h's in H under the constraints that are introduced in the first stage. The iterative process in which the arguments are chosen (that is, the first stage above) corresponds to the simulator's choice of the various block-prefixes $\overline{bp}$ (along with the indices i), on which h is applied.

At first glance, it seems obvious that the function $h^{(r)}$, which is uniformly distributed amongst all functions that are defined to be equal to h on all inputs (except for the input $(\eta, \overline{bp}_\xi)$, on which it equals r), is uniformly distributed in H. Taking a closer look, however, one realizes that a rigorous proof for the above claim is more complex than one may initially think, since it is not even clear that an h that is defined by the above process actually belongs to the family H.

The main difficulty in proving the above lies in the fact that the simulator's queries may "adaptively" depend on previous answers it has received (which, in turn, may depend on previous outcomes of h). The key observation used in order to overcome this difficulty is that, for *every* family of $t_S(n)$-wise independent functions and for *every* sequence of at most $t_S(n)$ arguments (and, in particular, for an adaptively chosen sequence), the values of a uniformly chosen function when applied to the arguments

in the sequence are uniformly and independently distributed. Thus, as long as the values assigned to the function in the first stage of the above process are uniformly and independently distributed (which is indeed the case, even if we constrain one output to be equal to r), the process will yield a uniformly distributed function from H. □

5.2. The success probability of the cheating prover. We start by introducing two important notions that will play a central role in the analysis of the success probability of the cheating prover.

5.2.1. Grouping queries according to their iteration-prefixes. In what follows, it will be convenient to group the queries of the simulator into different classes based on different iteration-prefixes. (Recall that the iteration-prefix of a query $\overline{q}$ (satisfying $\pi_{\mathrm{sn}}(\overline{q}) = (\ell, i)$ and $\pi_{\mathrm{msg}}(\overline{q}) = j > 1$) is the prefix of $\overline{q}$ that ends with the $(j-1)$st prover message in session (ℓ, n).) Grouping by iteration-prefixes particularly makes sense in the case in which two queries are of the same length (see discussion below). Nevertheless, by Definition 4.3, two queries may have the same iteration-prefix even if they are of *different* lengths (see below).

DEFINITION 5.3 (ip-different queries). *Two queries, $\overline{q}_1$ and $\overline{q}_2$ (of possibly different lengths), are said to be* ip-different *iff they have different iteration-prefixes (that is, $ip(\overline{q}_1) \neq ip(\overline{q}_2)$).*

By Definition 4.3, if two queries, $\overline{q}_1$ and $\overline{q}_2$, satisfy $ip(\overline{q}_1) = ip(\overline{q}_2)$, then the following two conditions must hold: (1) $\pi_{\mathrm{sn}}(\overline{q}_1) = (\ell, i_1)$, $\pi_{\mathrm{sn}}(\overline{q}_2) = (\ell, i_2)$, and (2) $\pi_{\mathrm{msg}}(\overline{q}_1) = \pi_{\mathrm{msg}}(\overline{q}_2)$. However, it is not necessarily true that $i_1 = i_2$. In particular, it may very well be the case that q_1, q_2 have different lengths (i.e., $i_1 \neq i_2$) but are *not* ip-different. (Note that, if $i_1 = i_2$, then q_1 and q_2 are of equal length.) Still, even if two queries are of the same length and have the same iteration-prefix, it is not necessarily true that they are equal, as they may be different at some message which occurs after their iteration-prefixes.

Motivating Definition 5.3. Recall that a necessary condition for the success of the cheating prover is that, for every j, once a $\mathtt{p}_{j-1}^{(\eta)}$ message has been forwarded to the verifier (in step 4b), all subsequent $\mathtt{p}_{j-1}^{(\eta)}$ messages (that are not answered with ABORT) are equal to the forwarded message. In order to satisfy the above condition, it is sufficient to require that the cheating prover never reach steps 4b and 4c with two ip-different queries of equal length. The reason for this is that, if two queries of the same length have the same iteration-prefix, then they contain the *same* sequence of prover messages for the corresponding session (since all such messages are contained in the iteration-prefix), and so they agree on their $\mathtt{p}_{j-1}^{(\eta)}$ message. In particular, once a $\mathtt{p}_{j-1}^{(\eta)}$ message has been forwarded to the verifier (in step 4b), all subsequent queries that reach step 4c and are of the same length will have the same $\mathtt{p}_{j-1}^{(\eta)}$ messages as the first such query (since they have the same iteration-prefix).

In light of the above discussion, it is only natural to require that the number of ip-different queries that reach step 4c of the cheating prover be exactly one (as, in such a case, the above necessary condition is indeed satisfied).[22] Jumping ahead, we comment that the smaller the number of ip-different queries that correspond to block-

[22] In order to ensure the cheating prover's success, the above requirement should be augmented by the condition that session $(\ell^{(\overline{bp}_\xi)}, \eta)$ is accepted by $V_{g,h^{(r)}}$.

prefix $\overline{bp}_\xi$ is the smaller the probability that more than one ip-different query reaches step 4c is. The reason for this lies in the fact that the number of ip-different queries that correspond to block-prefix $\overline{bp}_\xi$ is equal to the number of different iteration-prefixes that correspond to $\overline{bp}_\xi$. In particular, the smaller the number of such iteration-prefixes is the smaller the probability that g will evaluate to 1 on more than a single iteration-prefix is (thus reaching step 4c with more than one ip-different query).

5.2.2. Useful block-prefixes. The probability that the cheating prover makes the honest verifier accept will be lower bounded by the probability that the ξth distinct block-prefix in $\text{EXEC}_x(\sigma, g, h)$ is η-useful (in the sense hinted above and defined next).

DEFINITION 5.4 (useful block-prefix). *A specific block-prefix* $\overline{bp} = (b_1, a_1, \ldots, b_\gamma, a_\gamma)$, *appearing in* $\text{EXEC}_x(\sigma, g, h)$, *is called* i-useful *if it satisfies the following two conditions:*

1. *For every* $j \in \{2, \ldots, k+1\}$, *the number of* ip*-different queries* $\overline{q}$ *in* $\text{EXEC}_x(\sigma, g, h)$ *that correspond to block-prefix* $\overline{bp}$ *and satisfy* $\pi_{\text{sn}}(\overline{q}) = (\ell^{(\overline{bp})}, i)$, $\pi_{\text{msg}}(\overline{q}) = j$, *and* $g(i, ip(\overline{q})) = 1$ *is exactly one.*
2. *The (only) query* $\overline{q}$ *in* $\text{EXEC}_x(\sigma, g, h)$ *that corresponds to block-prefix* $\overline{bp}$ *and that satisfies* $\pi_{\text{sn}}(\overline{q}) = (\ell^{(\overline{bp})}, i)$, $\pi_{\text{msg}}(\overline{q}) = k + 1$, *and* $g(i, ip(\overline{q})) = 1$ *is answered with* ACCEPT *by* $V_{g,h}$.

If there exists an $i \in \{1, \ldots, n\}$ *so that a block-prefix is* i*-useful, then this block-prefix is called* useful.

Condition 1 in Definition 5.4 states that, for every fixed value of j, there exists exactly one iteration-prefix, $\overline{ip}$, that corresponds to queries of the block-prefix $\overline{bp}$ and the jth message so that $g(i, \overline{ip})$ evaluates to 1. Condition 2 asserts that the last verifier message in the ith main session of recursive block number $\ell = \ell^{(\overline{bp})}$ is equal to ACCEPT. It follows that, if the cheating prover happens to select $(\sigma, g, h, \xi, \eta)$ so that block-prefix $\overline{bp}_\xi$ (i.e., the ξth distinct block-prefix in $\text{EXEC}_x(\sigma, g, h^{(r)})$) is η-useful, then it convinces $V(x, r)$. The reason is that (by condition 2) the last message in session $(\ell^{(\overline{bp}_\xi)}, \eta)$ is answered with ACCEPT[23] and that (by condition 1) the emulation does not get into trouble in step 4c of the cheating prover. (To see this, notice that each prover message in session $(\ell^{(\overline{bp}_\xi)}, \eta)$ will end up reaching step 4c only once.)

Let $\langle P^*, V\rangle(x) = \langle P^*(\sigma, g, h, \xi, \eta), V(r)\rangle(x)$ denote the random variable representing the (local) output of the honest verifier V when interacting with the cheating prover P^* on common input x, where σ, g, h, ξ, η are the initial random choices made by the cheating prover P^* and r is the randomness used by the honest verifier V. Adopting this notation, we will say that the cheating prover $P^* = P^*(x, \sigma, g, h, \xi, \eta)$ has convinced the honest verifier $V = V(x, r)$ if $\langle P^*, V\rangle(x) = \text{ACCEPT}$. With this notation, we are ready to formalize the above discussion.

CLAIM 5.5. *If the cheating prover happens to select* $(\sigma, g, h, \xi, \eta)$ *so that the* ξ*th distinct block-prefix in* $\text{EXEC}_x(\sigma, g, h^{(r)})$ *is* η*-useful, then the cheating prover convinces* $V(x, r)$ *(i.e.,* $\langle P^*, V\rangle(x) = \text{ACCEPT}$*).*

Proof. Let us fix $x \in \{0,1\}^n$, $\sigma \in \{0,1\}^*$, $g \in G$, $h \in H$, $r \in \{1, \ldots, \rho_V(n)\}$, $\eta \in \{1, \ldots, n\}$, and $\xi \in \{1, \ldots, t_S(n)\}$. We show that, if the ξth distinct block-prefix in $\text{EXEC}_x(\sigma, g, h^{(r)})$ is η-useful, then the cheating prover $P^*(x, \sigma, g, h, \xi, \eta)$ convinces the honest verifier $V(x, r)$.

[23] Notice that $V(x, r)$ behaves exactly as $V_{g,h^{(r)}}$ behaves on queries that correspond to the ξth distinct iteration-prefix in $\text{EXEC}_x(\sigma, g, h^{(r)})$.

By the definition of the cheating prover, the prover messages that are actually forwarded to the honest verifier (in step 4b) correspond to session $(\ell^{(\overline{bp}_\xi)}, \eta)$. Specifically, messages that are forwarded by the cheating prover are of the form $\mathtt{p}_{j-1}^{(\eta)}$ and correspond to queries $\overline{q}$ that satisfy $\pi_{\mathrm{sn}}(\overline{q}) = (\ell^{(\overline{bp}_\xi)}, \eta)$, $\pi_{\mathrm{msg}}(\overline{q}) = j$, and $g(\eta, ip(\overline{q})) = 1$. Since the ξth distinct block-prefix in $\mathrm{EXEC}_x(\sigma, g, h^{(r)})$ is η-useful, we have that, for every $j \in \{2, \ldots, k+1\}$, there is exactly one query $\overline{q}$ that satisfies the above conditions. Thus, for every $j \in \{2, \ldots, k+1\}$, the cheating prover never reaches step 4c with two different $\mathtt{p}_{j-1}^{(\eta)}$ messages. Here we use the fact that, if two queries of the same length are not ip-different (i.e., have the same iteration-prefix), then the answers given by $V_{g,h^{(r)}}$ to these queries are identical (see the discussion above). This, in particular, means that P^* is answering the simulator's queries with the answers that would have been given by $V^{g,h^{(r)}}$ itself. (In other words, whenever the ξth distinct block-prefix in $\mathrm{EXEC}_x(\sigma, g, h^{(r)})$ is η-useful, the emulation does not "get into trouble" in step 4c of the cheating prover.)

At this point, we have that the cheating prover never fails to perform step 4c, and so the interaction that it is conducting with $V(x, r)$ "safely" reaches the $(k+1)$st verifier message in the protocol. To complete the proof, we have to show that, at the end of the interaction with the cheating-prover, $V(x, r)$ outputs `ACCEPT`. This is true since, by condition 2 of Definition 5.4, the query $\overline{q}$ that corresponds to block-prefix $\overline{bp}_\xi$ and satisfies $\pi_{\mathrm{sn}}(\overline{q}) = (\ell^{(\overline{bp}_\xi)}, \eta)$, $\pi_{\mathrm{msg}}(\overline{q}) = j$, and $g(\eta, ip(\overline{q})) = 1$, is answered with `ACCEPT`. Here we use the fact that $V(x, r)$ behaves exactly as $V_{g,h^{(r)}}$ behaves on queries that correspond to the ξth distinct block-prefix in $\mathrm{EXEC}_x(\sigma, g, h^{(r)})$. □

5.2.3. Reduction to rareness of legal transcripts without useful block-prefixes. Lemma 5.6 establishes the connection between the success probability of the simulator and the success probability of the cheating prover. Loosely speaking, the lemma asserts that, if S outputs a legal transcript with nonnegligible probability, then the cheating prover will succeed in convincing the honest verifier with nonnegligible probability. Since this is in contradiction to the computational soundness of the proof-system, we have that Lemma 5.6 actually implies the correctness of Lemma 4.5. (Recall that the contradiction hypothesis of Lemma 4.5 is that the probability that the simulator outputs a legal transcript is nonnegligible.)

LEMMA 5.6. *Suppose that* $\Pr_{\sigma,g,h}[(\sigma, g, h) \in \mathtt{AC}] > 1/p(n)$ *for some fixed polynomial* $p(\cdot)$. *Then the probability* (*taken over* $\sigma, g, h, \xi, \eta, r$) *that* $\langle P^*, V\rangle(x) = \mathtt{ACCEPT}$ *is at least* $\frac{1}{2 \cdot p(n) \cdot t_S(n) \cdot n}$.

Proof. Define $\mathsf{useful}_{\xi,\eta}(\sigma, g, h)$ to be true iff the ξth distinct block-prefix in $\mathrm{EXEC}_x(\sigma, g, h)$ is η-useful. Using Claim 5.5, we have

$$\Pr_{\sigma,g,h,\xi,\eta,r}\left[\langle P^*, V\rangle(x) = \mathtt{ACCEPT}\right] \geq \Pr_{\sigma,g,h,\xi,\eta,r}[\mathsf{useful}_{\xi,\eta}(\sigma, g, h^{(r)})], \tag{5.1}$$

where the second probability refers to an interaction between S and $V_{g,h^{(r)}}$. Since, for every value of σ, g, η, and ξ, when h and r are uniformly selected, the function $h^{(r)}$ is uniformly distributed (see Claim 5.2), we infer that

$$\Pr_{\sigma,g,h,\xi,\eta,r}[\mathsf{useful}_{\xi,\eta}(\sigma, g, h^{(r)})] = \Pr_{\sigma,g,h',\xi,\eta}[\mathsf{useful}_{\xi,\eta}(\sigma, g, h')]. \tag{5.2}$$

On the other hand, since ξ and η are distributed independently of (σ, g, h), we have

$$\begin{aligned}
\Pr_{\sigma,g,h,\xi,\eta}[\mathsf{useful}_{\xi,\eta}(\sigma,g,h)] &= \sum_{d=1}^{t_S(n)}\sum_{i=1}^{n}\Pr_{\sigma,g,h,\xi,\eta}[\mathsf{useful}_{d,i}(\sigma,g,h)\ \&\ (\xi=d\ \&\ \eta=i)] \\
&= \sum_{d=1}^{t_S(n)}\sum_{i=1}^{n}\Pr_{\sigma,g,h}[\mathsf{useful}_{d,i}(\sigma,g,h)]\cdot\Pr_{\xi,\eta}[\xi=d\ \&\ \eta=i] \\
&= \sum_{d=1}^{t_S(n)}\sum_{i=1}^{n}\Pr_{\sigma,g,h}[\mathsf{useful}_{d,i}(\sigma,g,h)]\cdot\frac{1}{t_S(n)\cdot n} \\
&\geq \Pr_{\sigma,g,h}[\exists d,i \text{ s.t. } \mathsf{useful}_{d,i}(\sigma,g,h)]\cdot\frac{1}{t_S(n)\cdot n}, \qquad (5.3)
\end{aligned}$$

where $t_S(n)$ is the bound used by the cheating prover (for the number of distinct block-prefixes in $\text{EXEC}_x(\sigma, g, h)$). Combining (5.1), (5.2), and (5.3), we get

$$\begin{aligned}
&\Pr_{\sigma,g,h,\xi,\eta,r}[\langle P^*,V\rangle(x)=\texttt{ACCEPT}] \\
&\qquad \geq \Pr_{\sigma,g,h}[\exists d,i \text{ s.t. } \mathsf{useful}_{d,i}(\sigma,g,h)]\cdot\frac{1}{t_S(n)\cdot n}. \qquad (5.4)
\end{aligned}$$

Recall that, by our hypothesis, $\Pr[(\sigma, g, h) \in \texttt{AC}] > 1/p(n)$ for some fixed polynomial $p(\cdot)$. We can thus rewrite and lower bound the value of $\Pr_{\sigma,g,h}[\exists d, i \text{ s.t. } \mathsf{useful}_{d,i}(\sigma, g, h)]$ in the following way:

$$\begin{aligned}
&\Pr[\exists d,i \text{ s.t. } \mathsf{useful}_{d,i}(\sigma,g,h)] \\
&\quad = 1-\Pr[\forall d,i\ \neg\mathsf{useful}_{d,i}(\sigma,g,h)] \\
&\quad = 1-\Pr[(\forall d,i\ \neg\mathsf{useful}_{d,i}(\sigma,g,h))\,\&\,((\sigma,g,h)\notin\texttt{AC})] \\
&\qquad -\Pr[(\forall d,i\ \neg\mathsf{useful}_{d,i}(\sigma,g,h))\,\&\,((\sigma,g,h)\in\texttt{AC})] \\
&\quad \geq 1-\Pr[(\sigma,g,h)\notin\texttt{AC}]-\Pr[(\forall d,i\ \neg\mathsf{useful}_{d,i}(\sigma,g,h))\ \&\ (\sigma,g,h)\in\texttt{AC}] \\
&\quad > 1/p(n)-\Pr[(\forall d,i\ \neg\mathsf{useful}_{d,i}(\sigma,g,h))\ \&\ (\sigma,g,h)\in\texttt{AC}],
\end{aligned}$$

where all of the above probabilities are taken over (σ, g, h). It follows that, in order to show that $\Pr_{\sigma,g,h,\xi,\eta,r}[\langle P^*, V\rangle(x) = \texttt{ACCEPT}] > \frac{1}{2\cdot p(n)\cdot t_S(n)\cdot n}$, it will be sufficient to prove that, for every fixed polynomial $p'(\cdot)$, it holds that

$$\Pr_{\sigma,g,h}[(\forall d,i\ \neg\mathsf{useful}_{d,i}(\sigma,g,h))\ \&\ (\sigma,g,h)\in\texttt{AC}] \ < \ 1/p'(n).$$

Thus Lemma 5.6 is satisfied provided that $\Pr_{\sigma,g,h}[\forall d, i\ \neg\mathsf{useful}_{d,i}(\sigma, g, h)\ \&\ (\sigma, g, h) \in \texttt{AC}]$ is negligible. Consequently, Lemma 5.6 will follow by establishing Lemma 5.7.

LEMMA 5.7. *The probability (taken over σ, g, h) that, for all pairs (d, i), $\mathsf{useful}_{d,i}$ (σ, g, h) does not hold and that $(\sigma, g, h) \in \texttt{AC}$ is negligible. That is, the probability that $\text{EXEC}_x(\sigma, g, h)$ does not contain a useful block-prefix and S outputs a legal transcript is negligible.*

This completes the proof of Lemma 5.6. The rest of this section is devoted to proving Lemma 5.7. □

5.3. Proof of Lemma 5.7 (existence of useful block-prefixes in legal transcripts). The proof of Lemma 5.7 will proceed as follows. We first define a special kind of block-prefixes called *potentially useful* block-prefixes. Loosely speaking, these are block-prefixes in which the simulator does not make too many "rewinding" attempts. (Each "rewinding" corresponds to a different iteration-prefix.) Intuitively, the larger the number of "rewindings" is, the smaller the probability that a specific block-prefix is useful is. A block-prefix with a small number of "rewindings" is thus more likely to cause its block-prefix to be useful. Thus our basic approach will be to show the following.

1. In *every* "successful" execution (i.e., producing a legal transcript), the simulator generates a potentially useful block-prefix. This is proved by demonstrating, based on the structure of the schedule, that if no potentially useful block-prefix exists, then the simulation must take superpolynomial time.
2. Any potentially useful block-prefix is in fact useful with considerable probability. The argument that demonstrates this claim proceeds basically as follows. Consider a specific block-prefix $\overline{bp}$, let $\ell = \ell^{(\overline{bp})}$, and focus on a specific instance of session (ℓ, i) (that is, the specific instance of session (ℓ, i) that corresponds to block-prefix $\overline{bp}$). Suppose that block-prefix $\overline{bp}$ is potentially useful and that the above instance of session (ℓ, i) happens to be accepted by $V_{g,h}$. This means that there exist k queries with block-prefix $\overline{bp}$ that consist of the "main thread" that leads to acceptance (i.e., all queries that were not answered with `ABORT`). Recall that the decision to abort a session (ℓ, i) is made by applying the function g to i and the iteration-prefix of the corresponding query. Thus, if there are only a few different iteration-prefixes that correspond to block-prefix $\overline{bp}$ (which, as we said, is potentially useful), then there is considerable probability that *all* the queries having block-prefix $\overline{bp}$, but which do not belong to that "main thread," will be answered with `ABORT`. (That is, g will evaluate to 0 on the corresponding input.) If this lucky event occurs, then block-prefix $\overline{bp}$ will indeed be useful. (Recall that, for a block-prefix to be useful, we require that there exist a corresponding session that is accepted by $V_{g,h}$ and satisfies that for every $j \in \{2, \ldots, k+1\}$ there is a single iteration-prefix that makes g evaluate to 1 at the jth message of this session.)

Returning to the actual proof, we start by introducing the necessary definition (of a potentially useful block-prefix). Recall that, for any $g \in G$ and $h \in H$, the running time of the simulator S with oracle access to $V_{g,h}$ is bounded by $t_S(n)$. Let c be a constant such that $t_S(n) \leq n^c$ for all sufficiently large n.

DEFINITION 5.8 (potentially useful block-prefix). *A specific block-prefix* $\overline{bp} = (b_1, a_1, \ldots, b_\gamma, a_\gamma)$, *appearing in* $\text{EXEC}_x(\sigma, g, h)$, *is called* potentially useful *if it satisfies the following two conditions:*

1. *The number of* ip*-different queries that correspond to block-prefix* $\overline{bp}$ *is at most* k^{c+1}.
2. *The execution of the simulator reaches the end of the block that corresponds to block-prefix* $\overline{bp}$. *That is,* $\text{EXEC}_x(\sigma, g, h)$ *contains a query* $\overline{q}$ *that ends with the* $(k+1)$*st prover message in the* n*th main session of recursive block number* $\ell^{(\overline{bp})}$ (*i.e., some* $\mathsf{p}_{k+1}^{(\ell^{(\overline{bp})}, n)}$ *message*).

We stress that the bound k^{c+1} in condition 1 above refers to the same constant $c > 0$ that is used in the time bound $t_S(n) \leq n^c$. Using Definition 5.3 (of ip-different queries), we have that a bound of k^{c+1} on the number of ip-different queries that cor-

respond to block-prefix $\overline{bp}$ induces an upper bound on the total number of iteration-prefixes that correspond to block-prefix $\overline{bp}$. Note that this is in contrast to the definition of a useful block-prefix (Definition 5.4), in which we have a bound only on the number of ip-different queries of a specific length (i.e., the number of ip-different queries that correspond to a specific message in a specific session).

Turning to condition 2 of Definition 5.8, we recall that the query $\overline{q}$ ends with a $\mathsf{p}_{k+1}^{(\ell^{(\overline{bp})},n)}$ message (i.e., the last prover message of recursive block number $\ell^{(\overline{bp})}$). Technically speaking, this means that $\overline{q}$ does not actually correspond to block-prefix $\overline{bp}$ (since, by definition of the recursive schedule, the answer to query $\overline{q}$ is a message that does not belong to recursive block number $\ell^{(\overline{bp})}$). Nevertheless, since, before making query $\overline{q}$, the simulator has made queries to all prefixes of $\overline{q}$, we are guaranteed that, for every $i \in \{1,\ldots,n\}$ and $j \in \{1,\ldots,k+1\}$, the simulator has made a query $\overline{q}_{i,j}$ that is a prefix of $\overline{q}$, corresponds to block-prefix $\overline{bp}$, and satisfies $\pi_{\mathrm{sn}}(\overline{q}) = (\ell^{(\overline{bp})}, i)$ and $\pi_{\mathrm{msg}}(\overline{q}) = j$. (In other words, all messages of all sessions in recursive block number $\ell^{(\overline{bp})}$ have occurred during the execution of the simulator.) Furthermore, since the (modified) simulator does not make a query that is answered with a `DEVIATION` message (in step 1' of $V_{g,h}$) and it does make the query $\overline{q}$, we are guaranteed that the partial execution transcript induced by the query $\overline{q}$ contains the accepting conversations of at least $\frac{n^{1/2}}{4}$ sessions in recursive block number $\ell^{(\overline{bp})}$. (The latter observation will be used only at a later stage (while proving Lemma 5.7).)

It is worth noting that, whereas the definition of a useful block-prefix refers to the contents of iteration-prefixes (induced by the queries) that are sent by the simulator, the definition of a potentially useful block-prefix refers only to their quantity (and neither to their contents nor to the effect of the application of g on them).[24] It is thus natural that statements referring to potentially useful block-prefixes tend to have a combinatorial flavor. The following lemma is no exception. It asserts that *every* "successful" execution of the simulator must contain a potentially useful block-prefix (or, otherwise, the simulator will run in superpolynomial time).

LEMMA 5.9. *For any* $(\sigma,g,h) \in \mathtt{AC}_x$, $\mathrm{EXEC}_x(\sigma,g,h)$ *contains a potentially useful block-prefix.*

5.3.1. Proof of Lemma 5.9 (existence of potentially useful block-prefixes). The proof of Lemma 5.9 is by contradiction. We assume the existence of a triplet $(\sigma,g,h) \in \mathtt{AC}$ so that every block-prefix in $\mathrm{EXEC}_x(\sigma,g,h)$ is not potentially useful, and we show that this implies that $S_\sigma^{V_h}(x)$ made strictly more than n^c queries (which contradicts the explicit hypothesis that the running time of S is bounded by n^c).

The query-and-answer tree. Throughout the proof of Lemma 5.9, we will fix an arbitrary $(\sigma,g,h) \in \mathtt{AC}$ as above and study the corresponding $\mathrm{EXEC}_x(\sigma,g,h)$. A key vehicle in this study is the notion of a *query-and-answer tree* introduced in [26] (and also used in [29]).[25] This is a rooted tree (corresponding to $\mathrm{EXEC}_x(\sigma,g,h)$) in which vertices are labeled with verifier messages and edges are labeled with prover messages. The root is labeled with the fixed verifier message initializing the first session and has outgoing edges corresponding to the prover's messages initializing this session. In

[24] In particular, whereas the definition of a useful block-prefix refers to the outcome of g on iteration-prefixes that correspond to the relevant block-prefix, the definition of a potentially useful block-prefix refers only to the number of ip-different queries that correspond to the block-prefix (ignoring the outcomes of g on the relevant iteration-prefixes).

[25] The query-and-answer tree should not be confused with the tree that is induced by the recursive schedule.

general, paths down the tree (i.e., from the root to some vertices) correspond to queries. The query associated with such a path is obtained by concatenating the labeling of the vertices and edges along the path in the order traversed. We stress that each vertex in the query-and-answer tree corresponds to a query actually made by the simulator.

The index of the verifier (resp., prover) message labeling a specific vertex (resp., edge) in the tree is completely determined by the level in which the vertex (resp., edge) lies. That is, all vertices (resp., edges) in the ωth level of the tree are labeled with the ωth verifier (resp., prover) message in the schedule (out of a total of $n^2 \cdot (k+1)$ scheduled messages). For example, if $\omega = n^2 \cdot (k+1)$, all vertices (resp., edges) at the ωth level (which is the lowest possible level in the tree) are labeled with $\mathrm{v}_{k+1}^{(n,n)}$ (resp., $\mathrm{p}_{k+1}^{(n,n)}$). The difference between "sibling" vertices in the same level of the tree lies in the difference in the labels of their incoming edges (as induced by the simulator's "rewindings"). Specifically, whenever the simulator "rewinds" the interaction to the ωth verifier message in the schedule (i.e., makes a new query that is answered with the ωth verifier message), the corresponding vertex in the tree (which lies at the ωth level) will have multiple descendants one level down in the tree (i.e., at the $(\omega+1)$st level). The edges to each one of these descendants will be labeled with a different prover message.[26] We stress that the difference between these prover messages lies in the contents of the corresponding message (and not in its index).

By the above discussion, the outdegree of every vertex in the query-and-answer tree corresponds to the number of "rewindings" that the simulator has made to the relevant point in the schedule. (The order in which the outgoing edges appear in the tree does not necessarily correspond to the order in which the "rewindings" were actually performed by the simulator.) Vertices in which the simulator does not perform a "rewinding" will thus have a single outgoing edge. In particular, in case that the simulator follows the prescribed prover strategy P (sending each scheduled message exactly once), all vertices in the tree will have outdegree one, and the tree will actually consist of a single path of total length $n^2 \cdot (k+1)$ (ending with an edge that is labeled with a $\mathrm{p}_{k+1}^{(n,n)}$ message).

Recall that, by our conventions regarding the simulator, before making a query $\overline{q}$, the simulator has made queries to all prefixes of $\overline{q}$. Since every query corresponds to a path down the tree, we have that every particular path down the query-and-answer tree is developed from the root downward. (That is, within a specific path, a level $\omega < \omega'$ vertex is always visited before a level ω' vertex.) However, we cannot say anything about the order in which *different* paths in the tree are developed. (For example, we cannot assume that the simulator has made all queries that end at a level ω vertex before making any other query that ends at a level $\omega' > \omega$ vertex or that it has visited all vertices of level ω in some specific order.) To summarize, the only guarantee that we have about the order in which the query-and-answer tree is developed is implied by the convention that, before making a specific query, the simulator has made queries to all relevant prefixes.

Satisfied path. A path from one node in the tree to some of its descendants is said to *satisfy session* i if the path contains edges (resp., vertices) for each of the messages sent by the prover (resp., verifier) in session i. A path is called *satisfied* if

[26] In particular, the shape of the query-and-answer tree is completely determined by the contents of prover messages in $\mathrm{EXEC}_x(\sigma, g, h)$ (whereas the contents of verifier answers given by $V_{g,h}$ have no effect on the shape of the tree).

it satisfies all sessions for which the verifier's first message appears along the path. One important example for a satisfied path is the path that starts at the root of the query-and-answer tree and ends with an edge that is labeled with a $\mathtt{p}_{k+1}^{(n,n)}$ message. This path contains all $n^2 \cdot (k+1)$ messages in the schedule (and so satisfies all n^2 sessions in the schedule). We stress that the contents of messages (occurring as labels) along a path are completely irrelevant to the question of whether the path is satisfied or not. In particular, a path may be satisfied even if some (or even all) of the vertices along it are labeled with `ABORT`.

Recall that, by our conventions, the simulator never makes a query that is answered with the `DEVIATION` message. We are thus guaranteed that, for every completed block along a path in the tree, at least $\frac{n^{1/2}}{4}$ sessions are accepted by $V_{g,h}$. In particular, the vertices corresponding to messages of these accepted sessions cannot be labeled with `ABORT`.

Good subtree. Consider an arbitrary subtree (of the query-and-answer tree) that satisfies the following two conditions:

1. The subtree is rooted at a vertex corresponding to the first message of some session so that this session is the first main session of some recursive invocation of the schedule.
2. Each path in the subtree is truncated at the last message of the relevant recursive invocation.

The full tree (i.e., the tree rooted at the vertex labeled with the first message in the schedule) is indeed such a tree, but we will need to consider subtrees which correspond to m sessions in the recursive schedule construction (i.e., correspond to $\mathcal{R}_m$). We call such a subtree *m-good* if it contains a satisfied path starting at the root of the subtree. Since $(\sigma, g, h) \in \mathtt{AC}$, we have that the simulator has indeed produced a "legal" transcript as output. It follows that the full tree contains a path from the root to a leaf that contains vertices (resp., edges) for each of the messages sent by the verifier (resp., prover) in all n^2 sessions of the schedule (as otherwise the transcript $S_\sigma^{V_{g,h}}(x)$ would not have been legal). In other words, the full tree contains a satisfied path and is thus n^2-good.

Note that, by the definition of the recursive schedule, two m-good subtrees are always disjoint. On the other hand, if $m' < m$, it may be the case that an m'-good subtree is contained in another m-good subtree. As a matter of fact, since an m-good subtree contains all messages of all sessions in a recursive block corresponding to $\mathcal{R}_m$, then it must contain at least k disjoint $\frac{m-n}{k}$-good subtrees (i.e., that correspond to the k recursive invocations of $\mathcal{R}_{\frac{m-n}{k}}$ made by $\mathcal{R}_m$).

The next lemma (which can be viewed as the crux of the proof) states that, if the contradiction hypothesis of Lemma 5.9 is satisfied, then the number of disjoint $\frac{m-n}{k}$-good subtrees that are contained in an m-good subtree is actually considerably larger than k.

LEMMA 5.10. *Suppose that every block-prefix that appears in* $\text{EXEC}_x(\sigma, g, h)$ *is not potentially useful. Then, for every $m \geq n$, every m-good subtree contains at least k^{c+1} disjoint $\frac{m-n}{k}$-good subtrees.*

Denote by $W(m)$ the size of an m-good subtree. (That is, $W(m)$ actually represents the work performed by the simulator on m concurrent sessions in our fixed scheduling.) It follows (from Lemma 5.10) that any m-good subtree must satisfy

$$W(m) \geq \begin{cases} 1 & \text{if } m \leq n, \\ k^{c+1} \cdot W\left(\frac{m-n}{k}\right) & \text{if } m > n. \end{cases} \tag{5.5}$$

Since, for all but finitely many n, (5.5) solves to $W(n^2) > n^c$ (see Appendix B), and since every vertex in the query-and-answer tree corresponds to a query actually made by the simulator, it follows that the hypothesis that the simulator runs in time that is bounded by n^c (and hence the full n^2-good tree must have been of size at most n^c) is contradicted. Thus Lemma 5.9 will actually follow from Lemma 5.10.

Proof of Lemma 5.10. Let T be an arbitrary m-good subtree of the query-and-answer tree. Considering the m sessions corresponding to an m-good subtree, we focus on the n main sessions of this level of the recursive construction. Let B_T denote the recursive block to which the indices of these n sessions belong. A *T-query* is a query $\overline{q}$ whose corresponding path down the query-and-answer tree ends with a node that belongs to T (recall that every query $\overline{q}$ appearing in $\text{EXEC}_x(\sigma, g, h)$ corresponds to a path down the full tree) and that satisfies $\pi_{\text{sn}}(\overline{q}) \in B_T$.[27] We first claim that all T-queries $\overline{q}$ in $\text{EXEC}_x(\sigma, g, h)$ have the same block-prefix. This block-prefix corresponds to the path from the root of the full tree to the root of T and is denoted by $\overline{bp}_T$.

FACT 5.11. *All T-queries in $\text{EXEC}_x(\sigma, g, h)$ have the same block-prefix (denoted by $\overline{bp}_T$).*

Proof. Assume, toward contradiction, that there exist two different T-queries $\overline{q}_1, \overline{q}_2$ so that $bp(\overline{q}_1) \neq bp(\overline{q}_2)$. In particular, $bp(\overline{q}_1)$ and $bp(\overline{q}_2)$ must differ in a message that precedes the first message of the first main session in B_T. (Note that, if two block-prefixes are equal in all messages preceding the first message of the first session of the relevant block, then, by definition, they are equal.[28]) This means that the paths that correspond to $\overline{q}_1$ and $\overline{q}_2$ split from each other before they reach the root of T. (Remember that T is rooted at a node corresponding to the first main session of recursive block B_T.) However, this contradicts the fact that both paths that correspond to these queries end with a node in T, and the fact follows. □

Using the hypothesis that no block-prefix in $\text{EXEC}_x(\sigma, g, h)$ is potentially useful, we prove the following claim.

CLAIM 5.12. *Let T be an m-good subtree. Then the number of* ip*-different queries that correspond to block-prefix $\overline{bp}_T$ is at least k^{c+1}.*

Proof. Since all block-prefixes that appear in $\text{EXEC}_x(\sigma, g, h)$ are not potentially useful (by the hypothesis of Lemma 5.10), this holds as a special case for block-prefix $\overline{bp}_T$. Let $\ell = \ell^{(\overline{bp}_T)}$ be the index of the recursive block that corresponds to block-prefix $\overline{bp}_T$ in $\text{EXEC}_x(\sigma, g, h)$. Since block-prefix $\overline{bp}_T$ is not potentially useful, at least one of the two conditions of Definition 5.8 is violated. In other words, one of the following two conditions is satisfied:

1. The number of ip-different queries that correspond to block-prefix $\overline{bp}_T$ is at least k^{c+1}.
2. The execution of the simulator does not reach the end of the block that corresponds to block-prefix $\overline{bp}_T$ (i.e., there is no query in $\text{EXEC}_x(\sigma, g, h)$ that ends with a $\mathsf{p}_{k+1}^{(\ell,n)}$ message that corresponds to block-prefix $\overline{bp}_T$).

Now, since T is an m-good subtree, it must contain a satisfied path. Such a path starts at the root of T and satisfies all sessions whose first verifier message appears

[27] Note that queries $\overline{q}$ that satisfy $\pi_{\text{sn}}(\overline{q}) \in B_T$ do not necessarily correspond to a path that ends with a node in T (as $\text{EXEC}_x(\sigma, g, h)$ may contain a different subtree T' that satisfies $B_T = B_{T'}$). Also note that there exist queries $\overline{q}$ whose corresponding path ends with a node that belongs to T but that satisfy $\pi_{\text{sn}}(\overline{q}) \notin B_T$. This is so since T may also contain vertices that correspond to messages in sessions which are not main sessions of B_T (in particular, all sessions that belong to the lower-level recursive blocks that are invoked by block B_T).

[28] Recall that the index of the relevant block is determined by the length of the corresponding block-prefix.

along the path. The key observation is that every satisfied path that starts at the root of subtree T must satisfy all of the main sessions in B_T (to see this, notice that the first message of all main sessions in B_T will always appear along such a path), and so it contains all messages of all main sessions in recursive block B_T. In particular, the subtree T contains a path that starts at the root of T and ends with an edge that is labeled with the last prover message in session number (ℓ, n) (i.e., a $\mathtt{p}_{k+1}^{(\ell,n)}$ message). In other words, the execution of the simulator *does* reach the end of the block that corresponds to block-prefix $\overline{bp}_T$ (since, for the above path to exist, the simulator must have made a query that ends with a $\mathtt{p}_{k+1}^{(\ell,n)}$ message that corresponds to block-prefix $\overline{bp}_T$), and so condition 2 does not apply. Thus the only reason that may cause block-prefix $\overline{bp}_T$ not to be potentially useful is condition 1. We conclude that the number of ip-different queries that correspond to block-prefix $\overline{bp}_T$ is at least k^{c+1}, as required. □

The following claim establishes the connection between the number of ip-different queries that correspond to block-prefix $\overline{bp}_T$ and the number of $\frac{m-n}{k}$-good subtrees contained in T. Loosely speaking, this is achieved based on the following three observations: (1) Two queries are said to be ip-different iff they have different iteration-prefixes. (2) Every iteration-prefix is a block-prefix of some subschedule one level down in the recursive construction (consisting of $\frac{m-n}{k}$ sessions). (3) Every such distinct block-prefix yields a distinct $\frac{m-n}{k}$-good subtree.

CLAIM 5.13. *Let T be an m-good subtree. Then, for every pair of ip-different queries that correspond to block-prefix $\overline{bp}_T$, the subtree T contains two disjoint $\frac{m-n}{k}$-good subtrees.*

Once Claim 5.13 is proved, we can use it in conjunction with Claim 5.12 to infer that T contains at least k^{c+1} disjoint $\frac{m-n}{k}$-good subtrees.

Proof. Before we proceed with the proof of Claim 5.13, we introduce the notion of an *iteration-suffix* of a query $\overline{q}$. This is the suffix of $\overline{q}$ that starts at the ending point of the query's iteration-prefix. A key feature satisfied by an iteration-suffix of a query is that it contains all of the messages of all sessions belonging to some invocation of the schedule one level down in the recursive construction. (This follows directly from the structure of our fixed schedule.)

DEFINITION 5.14 (iteration-suffix). *The* iteration-suffix *of a query $\overline{q}$ (satisfying $j = \pi_{\mathrm{msg}}(\overline{q}) > 1$), denoted by $is(\overline{q})$, is the suffix of $\overline{q}$ that begins at the ending point of the iteration-prefix of query $\overline{q}$. That is, for $\overline{q} = (b_1, a_1, \ldots, a_t, b_t)$, if $ip(\overline{q}) = (b_1, a_1, \ldots, b_{\delta-1}, a_\delta)$, then $is(\overline{q}) = (a_\delta, b_{\delta+1}, \ldots, a_t, b_t)$.*[29]

Let $\overline{q}$ be a query, and let $(\ell, i) = \pi_{\mathrm{sn}}(\overline{q})$, $j = \pi_{\mathrm{msg}}(\overline{q})$. Let $\mathcal{P}(\overline{q})$ denote the path corresponding to query $\overline{q}$ in the query-and-answer tree. Let $\mathcal{P}_{ip}(\overline{q})$ denote the subpath of $\mathcal{P}(\overline{q})$ that corresponds to the iteration-prefix $ip(\overline{q})$ of $\overline{q}$, and let $\mathcal{P}_{is}(\overline{q})$ denote the subpath of $\mathcal{P}(\overline{q})$ that corresponds to the iteration-suffix $is(\overline{q})$ of $\overline{q}$. That is, the subpath $\mathcal{P}_{ip}(\overline{q})$ starts at the root of the full tree and ends at a $\mathtt{p}_{j-1}^{(\ell,n)}$ message, whereas the subpath $\mathcal{P}_{is}(\overline{q})$ starts at a $\mathtt{p}_{j-1}^{(\ell,n)}$ message and ends at a $\mathtt{v}_{j}^{(\ell,i)}$ message. (In particular, path $\mathcal{P}(\overline{q})$ can be obtained by concatenating $\mathcal{P}_{ip}(\overline{q})$ with $\mathcal{P}_{is}(\overline{q})$.[30])

[29] This means that a_δ is of the form $\mathtt{p}_{j-1}^{(\ell,n)}$, where $(\ell, i) = \pi_{\mathrm{sn}}(\overline{q})$ and $j = \pi_{\mathrm{msg}}(\overline{q})$.

[30] To be precise, one should delete from the resulting concatenation one of the two consecutive edges which are labeled with $a_\delta = \mathtt{p}_{j-1}^{(\ell,n)}$. (One edge is the last in $\mathcal{P}_{ip}(\overline{q})$, and the other edge is the first in $\mathcal{P}_{is}(\overline{q})$.)

FACT 5.15. *For every query* $\overline{q} \in \text{EXEC}_x(\sigma, g, h)$*, the subpath* $\mathcal{P}_{is}(\overline{q})$ *is satisfied. Moreover,*

1. *the subpath* $\mathcal{P}_{is}(\overline{q})$ *satisfies all* $\frac{m-n}{k}$ *sessions of a recursive invocation one level down in the recursive construction (i.e., corresponding to* $\mathcal{R}_{\frac{m-n}{k}}$*);*
2. *if* $\overline{q}$ *corresponds to block-prefix* $\overline{bp}_T$*, then the subpath* $\mathcal{P}_{is}(\overline{q})$ *is contained in* T*.*

Proof. Let $(\ell, i) = \pi_{\text{sn}}(\overline{q})$ and $j = \pi_{\text{msg}}(\overline{q})$. By the nature of our fixed scheduling, the vertex in which subpath $\mathcal{P}_{is}(\overline{q})$ begins precedes the first message of all (nested) sessions in the $(j-1)$st recursive invocation made by recursive block number ℓ (i.e., an instance of $\mathcal{R}_{\frac{m-n}{k}}$ which is invoked by $\mathcal{R}_m$). Since query $\overline{q}$ is answered with a $\mathtt{v}_j^{(\ell,i)}$ message, we have that the subpath $\mathcal{P}_{is}(\overline{q})$ eventually reaches a vertex labeled with $\mathtt{v}_j^{(\ell,i)}$. In particular, the subpath $\mathcal{P}_{is}(\overline{q})$ (starting at a $\mathtt{p}_{j-1}^{(\ell,n)}$ edge and ending at a $\mathtt{v}_j^{(\ell,i)}$ vertex) contains the first and last messages of each of the above (nested) sessions and so contains edges (resp., vertices) for each prover (resp., verifier) message in these sessions. By definition, this means that all of these (nested) sessions are satisfied by $\mathcal{P}_{is}(\overline{q})$. Since the above (nested) sessions are the only sessions whose first message appears along the subpath $\mathcal{P}_{is}(\overline{q})$, we have that $\mathcal{P}_{is}(\overline{q})$ is satisfied. To see that whenever $\overline{q}$ corresponds to block-prefix $\overline{bp}_T$ the subpath $\mathcal{P}_{is}(\overline{q})$ is contained in the subtree T, we observe that both its starting point (i.e., a $\mathtt{p}_{j-1}^{(\ell,n)}$ edge) and its ending point (i.e., a $\mathtt{v}_j^{(\ell,i)}$ vertex) are contained in T. □

FACT 5.16. *Let* $\overline{q}_1, \overline{q}_2$ *be two* ip*-different queries. Then* $\mathcal{P}_{is}(\overline{q}_1)$ *and* $\mathcal{P}_{is}(\overline{q}_2)$ *are disjoint.*

Proof. Let $\overline{q}_1$ and $\overline{q}_2$ be two ip-different queries, let $(\ell_1, i_1) = \pi_{\text{sn}}(\overline{q}_1)$, $(\ell_2, i_2) = \pi_{\text{sn}}(\overline{q}_2)$, and let $j_1 = \pi_{\text{msg}}(\overline{q}_1)$, $j_2 = \pi_{\text{msg}}(\overline{q}_2)$. Recall that queries $\overline{q}_1$ and $\overline{q}_2$ are said to be ip-different iff they have different iteration-prefixes. Since $\overline{q}_1$ and $\overline{q}_2$ are assumed to be ip-different, so are iteration-prefixes $ip(\overline{q}_1)$ and $ip(\overline{q}_2)$. In particular, the paths $\mathcal{P}_{ip}(\overline{q}_1)$ and $\mathcal{P}_{ip}(\overline{q}_2)$ are different. We distinguish between the following two cases:

1. *Path* $\mathcal{P}_{ip}(\overline{q}_1)$ *splits from* $\mathcal{P}_{ip}(\overline{q}_2)$. In such a case, the ending points of paths $\mathcal{P}_{ip}(\overline{q}_1)$ and $\mathcal{P}_{ip}(\overline{q}_2)$ must belong to different subtrees of the query-and-answer tree. Since the starting point of an iteration-suffix is the ending point of the corresponding iteration-prefix, we must have that paths $\mathcal{P}_{is}(\overline{q}_1)$ and $\mathcal{P}_{is}(\overline{q}_2)$ are disjoint.
2. *Path* $\mathcal{P}_{ip}(\overline{q}_1)$ *is a prefix of path* $\mathcal{P}_{ip}(\overline{q}_2)$. That is, both $\mathcal{P}_{ip}(\overline{q}_1)$ and $\mathcal{P}_{ip}(\overline{q}_2)$ reach a $\mathtt{v}_{j_1-1}^{(\ell_1,n)}$ vertex, while path $\mathcal{P}_{ip}(\overline{q}_2)$ continues down the tree and reaches a $\mathtt{v}_{j_2-1}^{(\ell_2,n)}$ vertex. The key observation in this case is that either ℓ_1 is strictly smaller than ℓ_2 or j_1 is strictly smaller than j_2. The reason for this is that, in the case that both $\ell_1 = \ell_2$ and $j_1 = j_2$ hold, iteration-prefix $ip(\overline{q}_1)$ must be equal to iteration-prefix $ip(\overline{q}_2)$,[31] in contradiction to our hypothesis. Since path $\mathcal{P}_{is}(\overline{q}_1)$ starts at a $\mathtt{p}_{j_1-1}^{(\ell_1,n)}$ vertex and ends with a $\mathtt{v}_{j_1}^{(\ell_1,i_1)}$ vertex, and since path $\mathcal{P}_{is}(\overline{q}_2)$ starts with a $\mathtt{p}_{j_2-1}^{(\ell_2,n)}$ vertex, we have that the ending point of path $\mathcal{P}_{is}(\overline{q}_1)$ precedes the starting point of path $\mathcal{P}_{is}(\overline{q}_2)$. (This is so since, if $j_1 < j_2$, the $\mathtt{p}_{j_1}^{(\ell_1,i_1)}$ message will always precede/equal the $\mathtt{p}_{j_2-1}^{(\ell_2,n)}$ message.) In particular, paths $\mathcal{P}_{is}(\overline{q}_1)$ and $\mathcal{P}_{is}(\overline{q}_2)$ are disjoint.

[31] That is, unless $bp(\overline{q}_1) \neq bp(\overline{q}_2)$. However, in such a case, paths $\mathcal{P}_{ip}(\overline{q}_1)$ and $\mathcal{P}_{ip}(\overline{q}_2)$ must split from each other (since they differ in some message that belongs to their block-prefix), and we are back to case 1.

It follows that, for every two ip-different queries, $\overline{q}_1$ and $\overline{q}_2$, subpaths $\mathcal{P}_{is}(\overline{q}_1)$ and $\mathcal{P}_{is}(\overline{q}_2)$ are disjoint, as required. □

Returning to the proof of Claim 5.13, let $\overline{q}_1$ and $\overline{q}_2$ be two ip-different queries that correspond to block-prefix $\overline{bp}_T$ (as guaranteed by the hypothesis of Claim 5.13), and let $\mathcal{P}_{is}(\overline{q}_1)$ and $\mathcal{P}_{is}(\overline{q}_2)$ be as above. Consider the two subtrees, T_1 and T_2, of T that are rooted at the starting point of subpaths $\mathcal{P}_{is}(\overline{q}_1)$ and $\mathcal{P}_{is}(\overline{q}_2)$, respectively. (Note that, by Fact 5.15, T_1 and T_2 are indeed subtrees of T.) By definition of our recursive schedule, T_1 and T_2 correspond to $\frac{m-n}{k}$ sessions one level down in the recursive construction (i.e., to an instance of $\mathcal{R}_{\frac{m-n}{k}}$). Using Fact 5.15, we infer that subpath $\mathcal{P}_{is}(\overline{q}_1)$ (resp., $\mathcal{P}_{is}(\overline{q}_2)$) contains all messages of all sessions in T_1 (resp., T_2), and so the subtree T_1 (resp., T_2) is $\frac{m-n}{k}$-good. In addition, since subpaths $\mathcal{P}_{is}(\overline{q}_1)$ and $\mathcal{P}_{is}(\overline{q}_2)$ are disjoint (by Fact 5.16) and since, by definition of an $\frac{m-n}{k}$-good tree, two different $\frac{m-n}{k}$-good trees are always disjoint, then T_1 and T_2 (which, being rooted at different vertices, must be different) are also disjoint. It follows that, for every pair of different queries that correspond to block-prefix $\overline{bp}_T$, the subtree T contains two disjoint $\frac{m-n}{k}$-good subtrees. □

We are finally ready to establish Lemma 5.10 (using Claims 5.12 and 5.13). By Claim 5.12, we have that the number of different queries that correspond to block-prefix $\overline{bp}_T$ is at least k^{c+1}. Since (by Claim 5.13), for every pair of different queries that correspond to block-prefix $\overline{bp}_T$, the subtree T contains two disjoint $\frac{m-n}{k}$-good subtrees, we infer that T contains a total of at least k^{c+1} disjoint $\frac{m-n}{k}$-good subtrees (corresponding to the (at least) k^{c+1} different queries mentioned above). Lemma 5.10 follows. □

5.3.2. Back to the proof of Lemma 5.7 (existence of useful block-prefixes). Once the correctness of Lemma 5.9 is established, we may proceed with the proof of Lemma 5.7. Let $x \in \{0,1\}^n$. We bound from above the probability, taken over the choices of $\sigma \in \{0,1\}^*, g \xleftarrow{R} G$, and $h \xleftarrow{R} H$, that $(\sigma, g, h) \in \mathtt{AC}$ and that, for all $d \in \{1, \ldots, t_S(n)\}$ and all $i \in \{1, \ldots, n\}$, the dth distinct block-prefix in $\text{EXEC}_x(\sigma, g, h)$ is not i-useful. Specifically, we would like to show that

$$\Pr_{\sigma,g,h}[(\forall d, i\ \neg\mathsf{useful}_{d,i}(\sigma, g, h))\ \&\ ((\sigma, g, h) \in \mathtt{AC})] \tag{5.6}$$

is negligible. Define a boolean indicator $\mathsf{pot}-\mathsf{use}_d(\sigma, g, h)$ to be true iff the dth distinct block-prefix in $\text{EXEC}_x(\sigma, g, h)$ is potentially useful. As proved in Lemma 5.9, for any $(\sigma, g, h) \in \mathtt{AC}$, there exists an index $d \in \{1, \ldots, t_S(n)\}$ so that the dth block-prefix in $\text{EXEC}_x(\sigma, g, h)$ is potentially useful. In other words, for every $(\sigma, g, h) \in \mathtt{AC}$, $\mathsf{pot}-\mathsf{use}_d(\sigma, g, h)$ holds for some value of d. Thus (5.6) is upper bounded by

$$\Pr_{\sigma,g,h}\left[\bigvee_{d=1}^{t_S(n)} \mathsf{pot}-\mathsf{use}_d(\sigma, g, h)\ \&\ (\forall i \in \{1, \ldots, n\}\ \neg\mathsf{useful}_{d,i}(\sigma, g, h))\right]. \tag{5.7}$$

Consider a specific $d \in \{1, \ldots, t_S(n)\}$ so that $\mathsf{pot}-\mathsf{use}_d(\sigma, g, h)$ is satisfied (i.e., the dth block prefix in $\text{EXEC}_x(\sigma, g, h)$ is potentially useful). By condition 2 in the definition of a potentially useful block-prefix (Definition 5.8), the execution of the simulator reaches the end of the corresponding block in the schedule. In other words, there exists a query $\overline{q} \in \text{EXEC}_x(\sigma, g, h)$ that ends with the $(k+1)$st prover message in the nth main session of recursive block number $\ell^{(\overline{bp}_d)}$, where $\overline{bp}_d$ denotes the dth distinct block-prefix in $\text{EXEC}_x(\sigma, g, h)$ and $\ell^{(\overline{bp}_d)}$ denotes the index of the recursive block that corresponds to block-prefix $\overline{bp}_d$ in $\text{EXEC}_x(\sigma, g, h)$. Since, by our convention and the

modification of the simulator, S never generates a query that is answered with a DEVIATION message, we have that the partial execution transcript induced by query $\overline{q}$ must contain the accepting conversations of at least $\frac{n^{1/2}}{4}$ main sessions in block number $\ell^{(\overline{bp}_d)}$ (as otherwise query $\overline{q}$ would have been answered with the DEVIATION message in step 1' of $V_{g,h}$).

Let $\overline{q}^{(\overline{bp}_d)} = \overline{q}^{(\overline{bp}_d)}(\sigma, g, h)$ denote the first query in $\text{EXEC}_x(\sigma, g, h)$ that is as above (i.e., that ends with the $(k+1)$st prover message in the nth main session of recursive block number $\ell^{(\overline{bp}_d)}$, where $\overline{bp}_d$ denotes the dth block-prefix appearing in $\text{EXEC}_x(\sigma, g, h)$).[32] Define an additional boolean indicator $\mathsf{accept}_{d,i}(\sigma, g, h)$ to be true iff query $\overline{q}^{(\overline{bp}_d)}$ contains an accepting conversation for session $(\ell^{(\overline{bp}_d)}, i)$. (That is, no prover message in session $(\ell^{(\overline{bp}_d)}, i)$ is answered with ABORT, and the last verifier message of this session equals ACCEPT.[33]) It follows that, for every $d \in \{1, \ldots, t_S(n)\}$ that satisfies $\mathsf{pot} - \mathsf{use}_d(\sigma, g, h)$ (as above), there exists a set $\mathcal{S} \subset \{1, \ldots, n\}$ of size $\frac{n^{1/2}}{4}$ such that $\mathsf{accept}_{d,i}(\sigma, g, h)$ holds for every $i \in \mathcal{S}$. Thus (5.7) is upper bounded by

$$\Pr_{\sigma,g,h}\left[\bigvee_{d=1}^{t_S(n)} \bigvee_{\substack{\mathcal{S}\subset\{1,\ldots,n\} \\ |\mathcal{S}|=\frac{n^{1/2}}{4}}} \big(\mathsf{pot} - \mathsf{use}_d(\sigma, g, h)\ \&\big(\forall i \in \mathcal{S},\ \neg\mathsf{useful}_{d,i}(\sigma, g, h)\right.$$

$$\left.\ \&\ \mathsf{accept}_{d,i}(\sigma, g, h)\big)\big)\right]. \tag{5.8}$$

Using the union bound, we upper bound (5.8) by

$$\sum_{d=1}^{t_S(n)} \sum_{\substack{\mathcal{S}\subset\{1,\ldots,n\} \\ |\mathcal{S}|=\frac{n^{1/2}}{4}}} \Pr_{\sigma,g,h}\left[\mathsf{pot} - \mathsf{use}_d(\sigma, g, h)\,\&\big(\forall i \in \mathcal{S},\ \neg\mathsf{useful}_{d,i}(\sigma, g, h)\right.$$

$$\left.\ \&\ \mathsf{accept}_{d,i}(\sigma, g, h)\big)\right]. \tag{5.9}$$

The last expression is upper bounded using the following lemma, which bounds the probability that a specific set of different sessions corresponding to the same (in index) potentially useful block-prefix are accepted (at the first time that the recursive block to which they belong is completed) but still do not turn it into a useful block-prefix. In fact, we prove something stronger.

[32] Since the simulator is allowed to feed $V_{g,h}$ with different queries of the same length, we have that the execution of the simulator may reach the end of the corresponding block more than once (and thus $\text{EXEC}_x(\sigma, g, h)$ may contain more than a single query that ends with the $(k+1)$st prover message in the nth main session of block number $\ell^{(\overline{bp}_d)}$). Since each time that the simulator reaches the end of the corresponding block the above set of accepted sessions may be different, we are not able to pinpoint a specific set of accepted sessions without explicitly specifying to which one of the above queries we are referring. We solve this problem by explicitly referring to the first query that satisfies the above conditions. (Note that, in our case, such a query is always guaranteed to exist.)

[33] Note that the second condition implies the first one. Namely, if the last verifier message of session $(\ell^{(\overline{bp}_d)}, i)$ equals ACCEPT, then no prover message in this session could have been answered with ABORT.

LEMMA 5.17. *For every $\sigma \in \{0,1\}^*$, every $h \in H$, every $d \in \{1,\ldots,t_S(n)\}$, and every set of indices $S \subset \{1,\ldots,n\}$ so that $|S| > k$,*

$$\Pr_g\left[\mathsf{pot}-\mathsf{use}_d(\sigma,g,h)\ \&\ \left(\forall i \in S,\ \neg\mathsf{useful}_{d,i}(\sigma,g,h)\ \&\ \mathsf{accept}_{d,i}(\sigma,g,h)\right)\right] < (n^{-\left(\frac{1}{2}+\frac{1}{4k}\right)})^{|S|}.$$

Proof. Let $x \in \{0,1\}^*$. Fix some $\sigma \in \{0,1\}^*$, $h \in H$, $d \in \{1,\ldots,t_S(n)\}$, and a set $S \subset \{1,\ldots,n\}$. Denote by $\overline{bp}_d = \overline{bp}_d(g)$ the dth distinct block-prefix in $\text{EXEC}_x(\sigma,h,g)$ and by $\ell^{(\overline{bp}_d)}$ the index of its corresponding recursive block in the schedule. We bound the probability, taken over the choice of $g \xleftarrow{R} G$, that, for all $i \in S$, block-prefix $\overline{bp}_d$ is not i-useful even though it is potentially useful, and, for all $i \in S$, the query $\overline{q}^{(\overline{bp}_d)}$ contains an accepting conversation for session $(\ell^{(\overline{bp}_d)}, i)$.

A technical problem resolved. In order to prove Lemma 5.17, we need to focus on the dth distinct block-prefix in $\text{EXEC}_x(\sigma,h,g)$ (denoted by $\overline{bp}_d$) and analyze the behavior of a uniformly chosen g when applied to the various iteration-prefixes that correspond to $\overline{bp}_d$. However, in trying to do so we encounter a technical problem. This problem is caused by the fact that the contents of block-prefix $\overline{bp}_d$ depend on g.[34] In particular, it does not make sense to analyze the behavior of a uniformly chosen g on iteration-prefixes that correspond to an "undetermined" block-prefix (since it is not possible to determine the iteration-prefixes that correspond to $\overline{bp}_d$ when $\overline{bp}_d$ itself is not determined). To overcome the above problem, we rely on the following observations:

1. Whenever σ, h, and d are fixed, the content of block-prefix $\overline{bp}_d$ is completely determined by the output of g on inputs that have occurred *before* $\overline{bp}_d$ has been reached (i.e., has appeared as a block-prefix of some query) for the first time.
2. All iteration-prefixes that correspond to block-prefix $\overline{bp}_d$ occur *after* $\overline{bp}_d$ has been reached for the first time.

It is thus possible to carry out the analysis by considering the output of g only on inputs that have occurred *after* $\overline{bp}_d$ has been determined. That is, fixing σ, h, and d, we distinguish between: (a) the outputs of g that have occurred *before* the dth distinct block-prefix in $\text{EXEC}_x(\sigma,g,h)$ (i.e., $\overline{bp}_d$) has been reached and (b) the outputs of g that have occurred *after* $\overline{bp}_d$ has been reached. For every possible outcome of (a), we will analyze the (probabilistic) behavior of g only over the outcomes of (b). (Recall that once (a)'s outcome has been determined, the identities (but not the contents) of all relevant prefixes are well defined.) Since for *every* possible outcome of (a) the analysis will hold, it will particularly hold over all choices of g.

More formally, consider the following (alternative) way of describing a uniformly chosen $g \in G$ (at least as far as $\text{EXEC}_x(\sigma,g,h)$ is concerned). Let g_1, g_2 be two $t_S(n)$-wise independent hash functions uniformly chosen from G, and let σ, h, d be as above. We define $g^{(g_1,g_2)} = g^{(\sigma,h,d,g_1,g_2)}$ to be uniformly distributed among the functions g' that satisfy the following conditions: the value of g', when applied to an input α that has occurred *before* $\overline{bp}_d$ has been reached (in $\text{EXEC}_x(\sigma,g,h)$), is equal to $g_1(\alpha)$, whereas the value of g', when applied to an input α that has occurred *after* $\overline{bp}_d$ has been reached, is equal to $g_2(\alpha)$.

[34] Clearly, the contents of queries that appear in $\text{EXEC}_x(\sigma,g,h)$ may depend on the choice of the hash function g. (This is because the simulator may dynamically adapt its queries depending on the outcome of g on iteration-prefixes of past queries.) As a consequence, the contents of $\overline{bp}_d = \overline{bp}_d(g)$ may vary together with the choice of g.

Similarly to the proof of Claim 5.2, it can be shown that, for every σ, h, d as above, if g_1 and g_2 are uniformly distributed, then so is $g^{(g_1,g_2)}$. In particular,

$$\begin{aligned}\Pr_g\left[\mathsf{pot}-\mathsf{use}_d(\sigma,g,h)\ \&\ \left(\forall i\in S,\ \neg\mathsf{useful}_{d,i}(\sigma,g,h)\ \&\ \mathsf{accept}_{d,i}(\sigma,g,h)\right)\right]\\ =\Pr_{g_1,g_2}\left[\mathsf{pot}-\mathsf{use}_d(\sigma,g^{(g_1,g_2)},h)\ \&\ \left(\forall i\in S,\ \neg\mathsf{useful}_{d,i}(\sigma,g^{(g_1,g_2)},h)\right.\right.\\ \left.\left.\&\ \mathsf{accept}_{d,i}(\sigma,g^{(g_1,g_2)},h)\right)\right].\end{aligned}$$

By fixing g_1 and then analyzing the behavior of a uniformly chosen g_2 on the relevant iteration-prefixes, we resolve the above technical problem. This is due to the following two reasons: (1) For every choice of σ, h, d and for *every* fixed value of g_1, the block-prefix $\overline{bp}_d$ is completely determined (and the corresponding iteration-prefixes are well defined). (2) Once $\overline{bp}_d$ has been reached, the outcome of $g^{(g_1,g_2)}$, when applied to the *relevant* iteration-prefixes, is completely determined by the choice of g_2. Thus all we need to show in order to prove Lemma 5.17 is that, for *every* choice of g_1, the value of

$$\begin{aligned}\Pr_{g_2}\left[\mathsf{pot}-\mathsf{use}_d(\sigma,g^{(g_1,g_2)},h)\,\&\left(\forall i\in S,\ \neg\mathsf{useful}_{d,i}(\sigma,g^{(g_1,g_2)},h)\right.\right.\\ \left.\left.\&\ \mathsf{accept}_{d,i}(\sigma,g^{(g_1,g_2)},h)\right)\right]\end{aligned}\tag{5.10}$$

is upper bounded by $(n^{-(1/2+1/4k)})^{|S|}$.

Back to the actual proof of Lemma 5.17. Consider the block-prefix $\overline{bp}_d$, as determined by the choices of σ, h, d, and g_1, and focus on the iteration-prefixes that correspond to $\overline{bp}_d$ in $\text{EXEC}_x(\sigma, g, h)$. We next analyze the implications of $\overline{bp}_d$ being not i-useful, even though it is potentially useful, and, for all $i \in S$, query $\overline{q}^{(\overline{bp}_d)}$ contains an accepting conversation for session $(\ell^{(\overline{bp}_d)}, i)$.

Claim 5.18. *Let $\sigma \in \{0,1\}^*$, $g \in G$, $h \in H$, $d \in \{1,\ldots,t_S(n)\}$, and $S \subset \{1,\ldots,n\}$. Suppose that the indicator*

$$\left(\mathsf{pot}-\mathsf{use}_d(\sigma,g,h)\ \&\ (\forall i\in S,\neg\mathsf{useful}_{d,i}(\sigma,g,h)\ \&\ \mathsf{accept}_{d,i}(\sigma,g,h))\right)$$

is true. Then the following hold.

1. *The number of different iteration-prefixes that correspond to block-prefix $\overline{bp}_d$ is at most k^{c+1}.*
2. *For every $j \in \{2,\ldots,k+1\}$, there exists an iteration-prefix $\overline{ip}_j$ (corresponding to block-prefix $\overline{bp}_d$) so that, for every $i \in S$, we have $g(i, \overline{ip}_j) = 1$.*
3. *For every $i \in S$, there exists an (additional) iteration-prefix $\overline{ip}^{(i)}$ (corresponding to block-prefix $\overline{bp}_d$) so that, for every $j \in \{2,\ldots,k+1\}$, we have $\overline{ip}^{(i)} \neq \overline{ip}_j$, and $g(i, \overline{ip}^{(i)}) = 1$.*

In accordance with the discussion above, Claim 5.18 will be invoked with $g = g^{(g_1,g_2)}$.

Proof. Loosely speaking, item 1 follows directly from the hypothesis that block-prefix $\overline{bp}_d$ is potentially useful. In order to prove item 2, we also use the hypothesis that, for all $i \in S$, query $\overline{q}^{(\overline{bp}_d)}$ contains an accepting conversation for session $(\ell^{(\overline{bp}_d)}, i)$, and, in order to prove item 3, we additionally use the hypothesis that, for all $i \in S$, block-prefix $\overline{bp}_d$ is not i-useful. Details follow.

Proof of Item 1. The hypothesis that block-prefix $\overline{bp}_d$ is potentially useful (i.e., $\mathsf{pot}-\mathsf{use}_d(\sigma,g,h)$ holds) implies that the number of iteration-prefixes that correspond to block-prefix $\overline{bp}_d$ is at most k^{c+1} (as otherwise, the number of ip-different queries that correspond to $\overline{bp}_d$ would have been greater than k^{c+1}).

Proof of Item 2. Let $i \in S$, and recall that $\mathsf{accept}_{d,i}(\sigma,g,h)$ holds. In particular, we have that query $\overline{q}^{(\overline{bp}_d)}$ (i.e., the first query in $\text{EXEC}_x(\sigma,g,h)$ that ends with the $(k+1)$st prover message in the nth main session of recursive block number $\ell^{(\overline{bp}_d)}$) contains an accepting conversation for session $(\ell^{(\overline{bp}_d)},i)$. That is, no prover message in session $(\ell^{(\overline{bp}_d)},i)$ is answered with ABORT, and the last verifier message of this session equals ACCEPT. Since, by our conventions regarding the simulator, before making query $\overline{q}^{(\overline{bp}_d)}$, the simulator has made queries to all relevant prefixes, then it must be the case that all prefixes of query $\overline{q}^{(\overline{bp}_d)}$ have previously occurred as queries in $\text{EXEC}_x(\sigma,g,h)$. In particular, for every $i \in S$ and for every $j \in \{2,\ldots,k+1\}$, the execution of the simulator must contain a query $\overline{q}_{i,j}$ that is a prefix of $\overline{q}^{(\overline{bp}_d)}$ and that satisfies $bp(\overline{q}_{i,j}) = \overline{bp}_d$, $\pi_{\text{sn}}(\overline{q}_{i,j}) = (\ell^{(\overline{bp}_d)},i)$, $\pi_{\text{msg}}(\overline{q}_{i,j}) = j$, and $g(i,ip(\overline{q}_{i,j})) = 1$. (If $g(i,ip(\overline{q}_{i,j}))$ would have been equal to 0, query $\overline{q}^{(\overline{bp}_d)}$ would have contained a prover message in session $(\ell^{(\overline{bp}_d)},i)$ that is answered with ABORT, in contradiction to the fact that $\mathsf{accept}_{d,i}(\sigma,g,h)$ holds.) Since, for every $j \in \{2,\ldots,k+1\}$ and for every $i_1,i_2 \in S$, we have that $ip(\overline{q}_{i_1,j}) = ip(\overline{q}_{i_2,j})$ (as queries $\overline{q}_{i,j}$ are all prefixes of $\overline{q}_\ell$ and $|ip(\overline{q}_{i_1,j})| = |ip(\overline{q}_{i_2,j})|$), we can set $\overline{ip}_j = ip(\overline{q}_{i,j})$. It follows that, for every $j \in \{2,\ldots,k+1\}$, iteration-prefix $\overline{ip}_j$ corresponds to block-prefix $\overline{bp}_d$ (as queries $\overline{q}_{i,j}$ all have block-prefix $\overline{bp}_d$), and, for every $i \in S$, we have that $g(i,\overline{ip}_j) = 1$.

Proof of Item 3. Let $i \in S$, and recall that, in addition to the fact that $\mathsf{accept}_{d,i}(\sigma,g,h)$ holds, we have that $\mathsf{useful}_{d,i}(\sigma,g,h)$ does not hold. Notice that the only reason for which $\mathsf{useful}_{d,i}(\sigma,g,h)$ can be false (i.e., the dth block-prefix is not i-useful), is that condition 1 in Definition 5.4 is violated by $\text{EXEC}_x(\sigma,g,h)$. (Recall that $\mathsf{accept}_{d,i}(\sigma,g,h)$ holds, and so condition 2 in Definition 5.4 is indeed satisfied by query $\overline{q}_{i,k+1}$ (as defined above). This query corresponds to block-prefix $\overline{bp}_d$, satisfies $\pi_{\text{sn}}(\overline{q}_{i,k+1}) = (\ell^{(\overline{bp}_d)},i)$, $\pi_{\text{msg}}(\overline{q}_{i,k+1}) = k+1$, and $g(i,ip(\overline{q}_{i,k+1})) = 1$, and is answered with ACCEPT.)

For condition 1 in Definition 5.4 to be violated, there must exist a $j \in \{2,\ldots,k+1\}$, with two ip-different queries, $\overline{q}_1$ and $\overline{q}_2$, that correspond to block-prefix $\overline{bp}_d$ and satisfy $\pi_{\text{sn}}(\overline{q}_1) = \pi_{\text{sn}}(\overline{q}_2) = (\ell^{(\overline{bp}_d)},i)$, $\pi_{\text{msg}}(\overline{q}_1) = \pi_{\text{msg}}(\overline{q}_2) = j$, and $g(i,ip(\overline{q}_1)) = g(i,ip(\overline{q}_2)) = 1$. Since, by definition, two queries are considered ip-different only if they differ in their iteration-prefixes, we have that there exist two different iteration-prefixes $\overline{ip}(\overline{q}_1)$ and $\overline{ip}(\overline{q}_2)$ (of the same length) that correspond to block-prefix $\overline{bp}_d$ and satisfy $g(i,\overline{ip}(\overline{q}_1)) = g(i,\overline{ip}(\overline{q}_2)) = 1$. Since iteration-prefixes $\overline{ip}_2,\ldots,\overline{ip}_{k+1}$ (from item 2 above) are all of distinct length, and since the only iteration-prefix in $\overline{ip}_2,\ldots,\overline{ip}_{k+1}$ that can be equal to either $\overline{ip}(\overline{q}_1)$ or $\overline{ip}(\overline{q}_2)$ is $\overline{ip}_j$ (note that this is the only iteration-prefix having the same length as $\overline{ip}(\overline{q}_1)$ and $\overline{ip}(\overline{q}_2)$), then it must be the case that at least one of $\overline{ip}(\overline{q}_1),\overline{ip}(\overline{q}_2)$ is different from all of $\overline{ip}_2,\ldots,\overline{ip}_{k+1}$. (Recall that $\overline{ip}(\overline{q}_1)$ and $\overline{ip}(\overline{q}_2)$ are different, which means

that they cannot both be equal to $\overline{ip}_j$.) In particular, for every $i \in S$ (that satisfies $\mathsf{useful}_{d,i}(\sigma, g, h)$ & $\mathsf{accept}_{d,i}(\sigma, g, h)$), there exists at least one (extra) iteration-prefix, $\overline{ip}^{(i)} \in \{\overline{ip}(\overline{q}_1), \overline{ip}(\overline{q}_2)\}$ that corresponds to block-prefix $\overline{bp}_d$, differs from $\overline{ip}_j$ for every $j \in \{2, \ldots, k+1\}$, and satisfies $g_2(i, \overline{ip}^{(i)}) = 1$.

This completes the proof of Claim 5.18. □

Recall that the hash function g_2 is chosen at random from a $t_S(n)$-wise independent family. Since, for every pair of different iteration-prefixes, the function g_2 will have different inputs, g_2 will have independent outputs when applied to different iteration-prefixes (since no more than $t_S(n)$ queries are made by the simulator). Similarly, for every pair of different $i, i' \in S$, g_2 will have different input and thus independent output. In other words, all outcomes of g_2 that are relevant to block-prefix $\overline{bp}_d$ are independent of each other. Since a uniformly chosen g_2 will output 1 with probability $n^{-1/2k}$, we may view every application of g_2 on iteration-prefixes that correspond to $\overline{bp}_d$ as an independently executed experiment that succeeds with probability $n^{-1/2k}$.[35]

Using item 1 of Claim 5.18, the applications of g_2 which are relevant to sessions $\{(\ell^{(\overline{bp}_d)}, i)\}_{i \in S}$ can be viewed as a sequence of at most k^{c+1} experiments (corresponding to at most k^{c+1} different iteration-prefixes). Each of these experiments consists of $|S|$ independent subexperiments (corresponding to the different $i \in S$), and each subexperiment succeeds with probability $n^{-1/2k}$. Item 2 of Claim 5.18 now implies that at least k of the above experiments will fully succeed (that is, all of their subexperiments will succeed), while item 3 implies that, for every $i \in S$, there exists an additional successful subexperiment (that is, a subexperiment of one of the $k^{c+1} - k$ remaining experiments). Using the fact that the probability that a subexperiment succeeds is $n^{-1/2k}$, we infer that the probability that an experiment fully succeeds is equal to $(n^{-1/2k})^{|S|}$. In particular, the probability in (5.10) is upper bounded by the probability that the following two events occur (these events correspond to items 2 and 3 of Claims 5.18, respectively):

Event 1. *In a sequence of* (*at most* k^{c+1}) *experiments, each succeeding with probability* $(n^{-1/2k})^{|S|}$, *there exist* k *successful experiments.* (The success probability corresponds to the probability that, for every $i \in S$, we have $g_2(i, \overline{ip}_j) = 1$ (see item 2 of Claim 5.18).)

Event 2. *For every one out of* $|S|$ *sequences of the remaining* (*at most* $k^{c+1} - k$) *subexperiments, each succeeding with probability* $n^{-1/2k}$, *there exists at least one successful experiment.* (In this case, the success probability corresponds to the probability that iteration-prefix $\overline{ip}^{(i)}$ satisfies $g_2(i, \overline{ip}^{(i)}) = 1$ (see item 3 of Claim 5.18).)

For $i \in |S|$ and $j \in [k^{c+1}]$, let us denote the success of the ith subexperiment in the jth experiment by $\chi_{i,j}$. By the above discussion for every i, j, the probability that $\chi_{i,j}$ holds is $n^{-1/2k}$ (independently of other $\chi_{i,j}$'s). We now have that, for Event 1 to succeed, there must exist a set of k experiments, $K \subseteq [k^{c+1}]$, so that, for all $(i, j) \in S \times K$, the event $\chi_{i,j}$ holds. For Event 2 to succeed, it must be the case that, for every $i \in S$, there exists one additional experiment (i.e., some $j \in [k^{c+1}] \setminus K$) so

[35] We may describe the process of picking $g_2 \xleftarrow{R} G$ as the process of independently letting the output of g_2 be equal to 1 with probability $n^{-1/2k}$ (each time a new input is introduced). Note that we will be doing so only for inputs that occur after block-prefix $\overline{bp}_d$ has been determined (as, in the above case, all inputs for g_2 are iteration-prefixes that correspond to block-prefix $\overline{bp}_d$, and such iteration-prefixes will occur only after $\overline{bp}_d$ has already been determined).

that $\chi_{i,j}$ holds. It follows that (5.10) is upper bounded by

$$\begin{aligned}
&\sum_{\substack{K\subseteq[k^{c+1}]\\|K|=k}} \Pr\left[\forall j\in K,\ \forall i\in S\ \text{ s.t. }\ \chi_{i,j}\right]\cdot\Pr\left[\forall i\in S,\ \exists j\in[k^{c+1}]\setminus K\ \text{ s.t. }\ \chi_{i,j}\right]\\
&\quad=\binom{k^{c+1}}{k}\cdot\left(\left(n^{-\frac{1}{2k}}\right)^{|S|}\right)^{k}\cdot\left(1-\left(1-n^{-\frac{1}{2k}}\right)^{k^{c+1}-k}\right)^{|S|}\\
&\quad<\left(k^{c+1}\right)^{k}\cdot\left(\left(n^{-\frac{1}{2k}}\right)^{|S|}\right)^{k}\cdot\left(k^{c+1}\cdot n^{-\frac{1}{2k}}\right)^{|S|} \qquad (5.11)\\
&\quad=\left(k^{c+1}\right)^{k+|S|}\cdot\left(n^{-\frac{1}{2k}}\right)^{k\cdot|S|+|S|}\\
&\quad=\left(k^{c+1}\right)^{k+|S|}\cdot\left(n^{-\frac{1}{4k}}\right)^{|S|}\left(n^{-\left(\frac{1}{2}+\frac{1}{4k}\right)}\right)^{|S|}\\
&\quad<\left(n^{-\left(\frac{1}{2}+\frac{1}{4k}\right)}\right)^{|S|}, \qquad (5.12)
\end{aligned}$$

where (5.11) holds whenever $k^{c+1}-k=o(n^{1/2k})$ (which is satisfied if $k=o(\frac{\log n}{\log\log n})$) and (5.12) holds whenever $(k^{c+1})^{k+|S|}\cdot(n^{-1/4k})^{|S|}<1$ (which is satisfied if both $|S|>k$ and $k=o(\frac{\log n}{\log\log n})$). This means that (5.10) is upper bounded by $(n^{-(1/2+1/4k)})^{|S|}$, and the proof of Lemma 5.17 is complete. □

Using Lemma 5.17, we upper bound (5.9) by

$$\begin{aligned}
t_S(n)\cdot\binom{n}{\frac{n^{1/2}}{4}}\cdot\left(n^{-\left(\frac{1}{2}+\frac{1}{4k}\right)}\right)^{\frac{n^{1/2}}{4}} &< t_S(n)\cdot\left(\frac{4\cdot e\cdot n}{n^{1/2}}\right)^{\frac{n^{1/2}}{4}}\cdot\left(n^{-\left(\frac{1}{2}+\frac{1}{4k}\right)}\right)^{\frac{n^{1/2}}{4}}\\
&= t_S(n)\cdot\left(\frac{4\cdot e}{n^{1/4k}}\right)^{\frac{n^{1/2}}{4}}\\
&< t_S(n)\cdot 2^{-\frac{n^{1/2}}{4}}, \qquad (5.13)
\end{aligned}$$

where inequality (5.13) holds whenever $8\cdot e<n^{1/4k}$ (which holds for $k<\frac{\log n}{4\cdot(3+\log e)}$). This completes the proof of Lemma 5.7 (since $\mathrm{poly}(n)\cdot 2^{-\Omega(n^{1/2})}$ is negligible).

Appendix A. Alternative description of the recursive schedule. The schedule consists of n^2 sessions. (Each session consists of $k+1$ prover messages and $k+1$ verifier messages.) It is defined recursively, where, for each $m\le n^2$, the schedule for sessions $i_1,\ldots,i_m$ (denoted $\mathcal{R}_{i_1,\ldots,i_m}$) proceeds as follows:

1. If $m\le n$, execute sessions $i_1,\ldots,i_m$ sequentially until they are all completed;
2. otherwise, for $j=1,\ldots,k+1$:
   (a) for $\ell=1,\ldots,n$:
      i. send the jth verifier message in session i_ℓ (i.e., $\mathtt{v}_j^{(i_\ell)}$);
      ii. send the jth prover message in session i_ℓ (i.e., $\mathtt{p}_j^{(i_\ell)}$);
   (b) if $j<k+1$, invoke a recursive copy of $\mathcal{R}_{i_{(n+(j-1)\cdot t+1)},\ldots,i_{(n+j\cdot t)}}$ (where $t\stackrel{\text{def}}{=}\lfloor\frac{m-n}{k}\rfloor$);
   (sessions $i_{(n+(j-1)\cdot t+1)},\ldots,i_{(n+j\cdot t)}$ are the next t remaining sessions out of $i_1,\ldots,i_m$).

Appendix B. Solving the recursion.

CLAIM B.1. *Suppose that* (5.5) *holds. Then, for all sufficiently large* n, $W(n^2) > n^c$.

Proof. By applying (5.5) iteratively $\log_k(n-1)$ times, we get

$$\begin{aligned} W(n^2) &\geq \left(k^{c+1}\right)^{\log_k(n-1)} \cdot W(n) \\ &\geq \left(k^{c+1}\right)^{\log_k(n-1)} \cdot 1 \\ &= (n-1)^{c+1} \\ &> n^c, \qquad \text{(B.1)} \end{aligned}$$

where (B.1) holds for all sufficiently large n. ∎

Acknowledgment. We are indebted to Oded Goldreich for his devoted help and technical contribution to this project.

REFERENCES

[1] N. ALON, L. BABAI, AND A. ITAI, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, J. Algorithms, 7 (1986), pp. 567–583.

[2] B. BARAK, *How to go beyond the black-box simulation barrier*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 106–115.

[3] M. BELLARE, R. IMPAGLIAZZO, AND M. NAOR, *Does parallel repetition lower the error in computationally sound protocols?*, in Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1997, pp. 374–383.

[4] M. BELLARE, S. MICALI, AND R. OSTROVSKY, *Perfect zero-knowledge in constant rounds*, in Proceedings of the 22nd ACM Symposium on Theory of Computing, ACM, New York, 1990, pp. 482–493.

[5] G. BRASSARD, D. CHAUM, AND C. CRÉPEAU, *Minimum disclosure proofs of knowledge*, J. Comput. System Sci., 37 (1988), pp.156–189.

[6] G. BRASSARD, C. CRÉPEAU, AND M. YUNG, *Constant-round perfect zero-knowledge computationally convincing protocols*, Theoret. Comput. Sci., 84 (1991), pp. 23–52.

[7] R. CANETTI, O. GOLDREICH, S. GOLDWASSER, AND S. MICALI, *Resettable zero-knowledge*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 235–244.

[8] R. CANETTI, J. KILIAN, E. PETRANK, AND A. ROSEN, *Black-box concurrent zero-knowledge requires* $\tilde{\Omega}(\log n)$ *rounds*, in Proceedings of the 33rd ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 570–579.

[9] M. N. WEGMAN AND J. L. CARTER, *New hash functions and their use in authentication and set equality*, J. Comput. System Sci., 22 (1981), pp. 265–279.

[10] B. CHOR AND O. GOLDREICH, *On the power of two-point based sampling*, J. Complexity, 5 (1989), pp. 96–106.

[11] I. DAMGÅRD, *Efficient concurrent zero-knowledge in the auxiliary string model*, in EuroCrypt2000, Lecture Notes in Comput. Sci. 1807, Springer-Verlag, New York, 2000, pp. 418–430.

[12] C. DWORK, M. NAOR, AND A. SAHAI, *Concurrent zero-knowledge*, in Proceedings of the 30th ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 409–418.

[13] C. DWORK AND A. SAHAI, *Concurrent zero-knowledge: Reducing the need for timing constraints*, in Crypto98, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, New York, 1998, pp. 442–457.

[14] U. FEIGE, *Alternative Models For Zero-Knowledge Interactive Proofs*, Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel, 1990.

[15] U. FEIGE AND A. SHAMIR, *Witness indistinguishability and witness hiding protocols*, in Proceedings of the 22nd ACM Symposium on Theory of Computing, ACM, New York, 1990, pp. 416–426.

[16] O. GOLDREICH, *Foundations of Cryptography—Basic Tools*, Cambridge University Press, Cambridge, UK, 2001.

[17] O. Goldreich and A. Kahan, *How to construct constant-round zero-knowledge proof systems for NP*, J. Cryptology, 9 (1996), pp. 167–189.
[18] O. Goldreich and H. Krawczyk, *On the composition of zero-knowledge proof systems*, SIAM J. Comput., 25 (1996), pp. 169–192.
[19] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*, J. ACM, 38 (1991), pp. 691–729.
[20] O. Goldreich and Y. Oren, *Definitions and properties of zero-knowledge proof systems*, J. Cryptology, 7 (1994), pp. 1–32.
[21] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems*, SIAM J. Comput., 18 (1989), pp. 186–208.
[22] S. Hada and T. Tanaka, *On the existence of* 3*-round zero-knowledge protocols*, in Crypto98, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, New York, pp. 408–423.
[23] J. Hastad, R. Impagliazzo, L.A. Levin, and M. Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput., 28 (1999), pp. 1364–1396.
[24] A. Joffe, *On a set of almost deterministic k-independent random variables*, Ann. Probab., 2 (1974), pp. 161–162.
[25] J. Kilian and E. Petrank, *Concurrent and resettable zero-knowledge in poly-logarithmic rounds*, in Proceedings of the 33rd ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 560–569.
[26] J. Kilian, E. Petrank, and C. Rackoff, *Lower bounds for zero-knowledge on the internet*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 484–492.
[27] M. Naor, *Bit commitment using pseudorandomness*, J. Cryptology, 4 (1991), pp. 151–158.
[28] R. Richardson and J. Kilian, *On the concurrent composition of zero-knowledge proofs*, in EuroCrypt99, Lecture Notes in Comput. Sci. 1592, Springer-Verlag, New York, 1999, pp. 415–431.
[29] A. Rosen, *A note on the round-complexity of concurrent zero-knowledge*, in Crypto2000, Lecture Notes in Comput. Sci. 1880, Springer-Verlag, New York, 2000, pp. 451–468.

SIAM J. COMPUT.
Vol. 32, No. 1, pp. 48–77

THE NONSTOCHASTIC MULTIARMED BANDIT PROBLEM*

PETER AUER†, NICOLÒ CESA-BIANCHI‡, YOAV FREUND§, AND ROBERT E. SCHAPIRE¶

Abstract. In the multiarmed bandit problem, a gambler must decide which arm of K nonidentical slot machines to play in a sequence of trials so as to maximize his reward. This classical problem has received much attention because of the simple model it provides of the trade-off between exploration (trying out each arm to find the best one) and exploitation (playing the arm believed to give the best payoff). Past solutions for the bandit problem have almost always relied on assumptions about the statistics of the slot machines.

In this work, we make no statistical assumptions whatsoever about the nature of the process generating the payoffs of the slot machines. We give a solution to the bandit problem in which an adversary, rather than a well-behaved stochastic process, has complete control over the payoffs. In a sequence of T plays, we prove that the per-round payoff of our algorithm approaches that of the best arm at the rate $O(T^{-1/2})$. We show by a matching lower bound that this is the best possible.

We also prove that our algorithm approaches the per-round payoff of *any* set of strategies at a similar rate: if the best strategy is chosen from a pool of N strategies, then our algorithm approaches the per-round payoff of the strategy at the rate $O((\log N)^{1/2}T^{-1/2})$. Finally, we apply our results to the problem of playing an unknown repeated matrix game. We show that our algorithm approaches the minimax payoff of the unknown game at the rate $O(T^{-1/2})$.

Key words. adversarial bandit problem, unknown matrix games

AMS subject classifications. 68Q32, 68T05, 91A20

PII. S0097539701398375

1. Introduction.

In the multiarmed bandit problem, originally proposed by Robbins [17], a gambler must choose which of K slot machines to play. At each time step, he pulls the arm of one of the machines and receives a reward or payoff (possibly zero or negative). The gambler's purpose is to maximize his return, i.e., the sum of the rewards he receives over a sequence of pulls. In this model, each arm is assumed to deliver rewards that are independently drawn from a fixed and unknown distribution. As reward distributions differ from arm to arm, the goal is to find the arm with the highest expected payoff as early as possible and then to keep gambling using that best arm.

The problem is a paradigmatic example of the trade-off between exploration and exploitation. On the one hand, if the gambler plays exclusively on the machine that he thinks is best ("exploitation"), he may fail to discover that one of the other arms actually has a higher expected payoff. On the other hand, if he spends too much time

*Received by the editors November 18, 2001; accepted for publication (in revised form) July 7, 2002; published electronically November 19, 2002. An early extended abstract of this paper appeared in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 1995, IEEE Computer Society, pp. 322–331.

http://www.siam.org/journals/sicomp/32-1/39837.html

†Institute for Theoretical Computer Science, Graz University of Technology, A-8010 Graz, Austria (pauer@igi.tu-graz.ac.at). This author gratefully acknowledges the support of ESPRIT Working Group EP 27150, Neural and Computational Learning II (NeuroCOLT II).

‡Department of Information Technology, University of Milan, I-26013 Crema, Italy (cesa-bianchi@dti.unimi.it). This author gratefully acknowledges the support of ESPRIT Working Group EP 27150, Neural and Computational Learning II (NeuroCOLT II).

§Banter Inc. and Hebrew University, Jerusalem, Israel (yoavf@cs.huji.ac.il).

¶AT&T Labs – Research, Shannon Laboratory, Florham Park, NJ 07932-0971 (schapire@research.att.com).

trying out all the machines and gathering statistics ("exploration"), he may fail to play the best arm often enough to get a high return.

The gambler's performance is typically measured in terms of "regret." This is the difference between the expected return of the optimal strategy (pulling consistently the best arm) and the gambler's expected return. Lai and Robbins proved that the gambler's regret over T pulls can be made, for $T \to \infty$, as small as $O(\ln T)$. Furthermore, they prove that this bound is optimal in the following sense: there is no strategy for the gambler with a better asymptotic performance.

Though this formulation of the bandit problem allows an elegant statistical treatment of the exploration-exploitation trade-off, it may not be adequate to model certain environments. As a motivating example, consider the task of repeatedly choosing a route for transmitting packets between two points in a communication network. To cast this scenario within the bandit problem, suppose there is a only a fixed number of possible routes and the transmission cost is reported back to the sender. Now, it is likely that the costs associated with each route cannot be modeled by a stationary distribution, so a more sophisticated set of statistical assumptions would be required. In general, it may be difficult or impossible to determine the right statistical assumptions for a given domain, and some domains may exhibit dependencies to an extent that no such assumptions are appropriate.

To provide a framework where one could model scenarios like the one sketched above, we present the adversarial bandit problem, a variant of the bandit problem in which *no* statistical assumptions are made about the generation of rewards. We assume only that each slot machine is initially assigned an arbitrary and unknown sequence of rewards, one for each time step, chosen from a bounded real interval. Each time the gambler pulls the arm of a slot machine, he receives the corresponding reward from the sequence assigned to that slot machine. To measure the gambler's performance in this setting we replace the notion of (statistical) regret with that of worst-case regret. Given any sequence $(j_1, \ldots, j_T)$ of pulls, where $T > 0$ is an arbitrary time horizon and each j_t is the index of an arm, the worst-case regret of a gambler for this sequence of pulls is the difference between the return the gambler would have had by pulling arms $j_1, \ldots, j_T$ and the actual gambler's return, where both returns are determined by the initial assignment of rewards. It is easy to see that, in this model, the gambler cannot keep his regret small (say, sublinear in T) for *all* sequences of pulls and with respect to the worst-case assignment of rewards to the arms. Thus, to make the problem feasible, we allow the regret to depend on the "hardness" of the sequence of pulls for which it is measured, where the hardness of a sequence is roughly the number of times one has to change the slot machine currently being played in order to pull the arms in the order given by the sequence. This trick allows us to effectively control the worst-case regret *simultaneously* for all sequences of pulls, even though (as one should expect) our regret bounds become trivial when the hardness of the sequence $(j_1, \ldots, j_T)$ we compete against gets too close to T.

As a remark, note that a deterministic bandit problem was also considered by Gittins [9] and Ishikida and Varaiya [13]. However, their version of the bandit problem is very different from ours: they assume that the player can compute ahead of time exactly what payoffs will be received from each arm, and their problem is thus one of optimization, rather than exploration and exploitation.

Our most general result is a very efficient, randomized player algorithm whose expected regret for any sequence of pulls is[1] $O(S\sqrt{KT\ln(KT)})$, where S is the hardness

[1]Though in this introduction we use the compact asymptotic notation, our bounds are proven for each finite T and almost always with explicit constants.

of the sequence (see Theorem 8.1 and Corollaries 8.2, 8.4). Note that this bound holds simultaneously for all sequences of pulls, for any assignments of rewards to the arms, and uniformly over the time horizon T. If the gambler is willing to impose an upper bound S on the hardness of the sequences of pulls for which he wants to measure his regret, an improved bound $O(\sqrt{SKT\ln(KT)})$ on the expected regret for these sequences can be proven (see Corollaries 8.3 and 8.5).

With the purpose of establishing connections with certain results in game theory, we also look at a special case of the worst-case regret, which we call "weak regret." Given a time horizon T, call "best arm" the arm that has the highest return (sum of assigned rewards) up to time T with respect to the initial assignment of rewards. The gambler's weak regret is the difference between the return of this best arm and the actual gambler's return. In the paper we introduce a randomized player algorithm, tailored to this notion of regret, whose expected weak regret is $O(\sqrt{KG_{\max}\ln K})$, where $G_{\max}$ is the return of the best arm—see Theorem 4.1 in section 4. As before, this bound holds for any assignments of rewards to the arms and uniformly over the choice of the time horizon T. Using a more complex player algorithm, we also prove that the weak regret is $O(\sqrt{KT\ln(KT/\delta)})$ with probability at least $1-\delta$ over the algorithm's randomization, for any fixed $\delta > 0$; see Theorems 6.3 and 6.4 in section 6. This also implies that, asymptotically for $T \to \infty$ and K constant, the weak regret is $O(\sqrt{T(\ln T)^{1+\varepsilon}})$ with probability 1 for any fixed $\varepsilon > 0$; see Corollary 6.5.

Our worst-case bounds may appear weaker than the bounds proved using statistical assumptions, such as those shown by Lai and Robbins [14] of the form $O(\ln T)$. However, when comparing our results to those in the statistics literature, it is important to point out an important difference in the asymptotic quantification. In the work of Lai and Robbins, the assumption is that the distribution of rewards that is associated with each arm is *fixed* as the total number of iterations T increases to infinity. In contrast, our bounds hold for any finite T, and, by the generality of our model, these bounds are applicable when the payoffs are randomly (or adversarially) chosen in a manner that does depend on T. It is this quantification order, and not the adversarial nature of our framework, which is the cause for the apparent gap. We prove this point in Theorem 5.1, where we show that, for *any* player algorithm for the K-armed bandit problem and for any T, there exists a set of K reward distributions such that the expected weak regret of the algorithm when playing on these arms for T time steps is $\Omega(\sqrt{KT})$.

So far we have considered notions of regret that compare the return of the gambler to the return of a sequence of pulls or to the return of the best arm. A further notion of regret which we explore is the regret for the best strategy in a given set of strategies that are available to the gambler. The notion of "strategy" generalizes that of "sequence of pulls": at each time step a strategy gives a recommendation, in the form of a probability distribution over the K arms, as to which arm to play next. Given an assignment of rewards to the arms and a set of N strategies for the gambler, call "best strategy" the strategy that yields the highest return with respect to this assignment. Then the regret for the best strategy is the difference between the return of this best strategy and the actual gambler's return. Using a randomized player that combines the choices of the N strategies (in the same vein as the algorithms for "prediction with expert advice" from [3]), we show that the expected regret for the best strategy is $O(\sqrt{KT\ln N})$—see Theorem 7.1. Note that the dependence on the number of strategies is only logarithmic, and therefore the bound is quite reasonable even when the player is combining a very large number of strategies.

The adversarial bandit problem is closely related to the problem of learning to

play an unknown N-person finite game, where the same game is played repeatedly by N players. A desirable property for a player is Hannan-consistency, which is similar to saying (in our bandit framework) that the weak regret per time step of the player converges to 0 with probability 1. Examples of Hannan-consistent player strategies have been provided by several authors in the past (see [5] for a survey of these results). By applying (slight extensions of) Theorems 6.3 and 6.4, we can provide an example of a simple Hannan-consistent player whose convergence rate is optimal up to logarithmic factors.

Our player algorithms are based in part on an algorithm presented by Freund and Schapire [6, 7], which in turn is a variant of Littlestone and Warmuth's [15] weighted majority algorithm and Vovk's [18] aggregating strategies. In the setting analyzed by Freund and Schapire, the player scores on each pull the reward of the chosen arm but gains access to the rewards associated with *all* of the arms (not just the one that was chosen).

2. Notation and terminology. An *adversarial bandit problem* is specified by the number K of possible actions, where each action is denoted by an integer $1 \leq i \leq K$, and by an *assignment of rewards*, i.e., an infinite sequence $\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots$ of vectors $\boldsymbol{x}(t) = (x_1(t), \ldots, x_K(t))$, where $x_i(t) \in [0,1]$ denotes the reward obtained if action i is chosen at time step (also called "trial") t. (Even though throughout the paper we will assume that all rewards belong to the $[0,1]$ interval, the generalization of our results to rewards in $[a,b]$ for arbitrary $a < b$ is straightforward.) We assume that the player knows the number K of actions. Furthermore, after each trial t, we assume the player knows only the rewards $x_{i_1}(1), \ldots, x_{i_t}(t)$ of the previously chosen actions $i_1, \ldots, i_t$. In this respect, we can view the player algorithm as a sequence $I_1, I_2, \ldots$, where each I_t is a mapping from the set $(\{1, \ldots, K\} \times [0,1])^{t-1}$ of action indices and previous rewards to the set of action indices.

For any reward assignment and for any $T > 0$, let

$$G_A(T) \stackrel{\text{def}}{=} \sum_{t=1}^{T} x_{i_t}(t)$$

be the *return at time horizon* T of algorithm A choosing actions $i_1, i_2, \ldots$. In what follows, we will write G_A instead of $G_A(T)$ whenever the value of T is clear from the context.

Our measure of performance for a player algorithm is the *worst-case regret*, and in this paper we explore variants of the notion of regret. Given any time horizon $T > 0$ and any sequence of actions $(j_1, \ldots, j_T)$, the (worst-case) regret of algorithm A for $(j_1, \ldots, j_T)$ is the difference

$$G_{(j_1,\ldots,j_T)} - G_A(T), \tag{1}$$

where

$$G_{(j_1,\ldots,j_T)} \stackrel{\text{def}}{=} \sum_{t=1}^{T} x_{j_t}(t)$$

is the return, at time horizon T, obtained by choosing actions $j_1, \ldots, j_T$. Hence, the regret (1) measures how much the player lost (or gained, depending on the sign of the difference) by following strategy A instead of choosing actions $j_1, \ldots, j_T$. A special

case of this is the regret of A for the best single action (which we will call *weak regret* for short), defined by

$$G_{\max}(T) - G_A(T),$$

where

$$G_{\max}(T) \stackrel{\text{def}}{=} \max_j \sum_{t=1}^{T} x_j(t)$$

is the return of the single globally best action at time horizon T. As before, we will write $G_{\max}$ instead of $G_{\max}(T)$ whenever the value of T is clear from the context.

As our player algorithms will be randomized, fixing a player algorithm defines a probability distribution over the set of all sequences of actions. All the probabilities $\mathbf{P}\{\cdot\}$ and expectations $\mathbf{E}[\cdot]$ considered in this paper will be taken with respect to this distribution.

In what follows, we will prove two kinds of bounds on the performance of a (randomized) player A. The first is a bound on the *expected regret*

$$G_{(j_1,\ldots,j_T)} - \mathbf{E}\left[G_A(T)\right]$$

of A for an arbitrary sequence $(j_1, \ldots, j_T)$ of actions. The second is a confidence bound on the weak regret. This has the form

$$\mathbf{P}\left\{G_{\max}(T) > G_A(T) + \varepsilon\right\} \leq \delta$$

and states that, with high probability, the return of A up to time T is not much smaller than that of the globally best action.

Finally, we remark that all of our bounds hold for *any* sequence $\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots$ of reward assignments, and most of them hold *uniformly* over the time horizon T (i.e., they hold for all T without requiring T as input parameter).

3. Upper bounds on the weak regret. In this section we present and analyze our simplest player algorithm, **Exp3** (which stands for "exponential-weight algorithm for exploration and exploitation"). We will show a bound on the expected regret of **Exp3** with respect to the single best action. In the next sections, we will greatly strengthen this result.

The algorithm **Exp3**, described in Figure 1, is a variant of the algorithm **Hedge** introduced by Freund and Schapire [6] for solving a different worst-case sequential allocation problem. On each time step t, **Exp3** draws an action i_t according to the distribution $p_1(t), \ldots, p_K(t)$. This distribution is a mixture of the uniform distribution and a distribution which assigns to each action a probability mass exponential in the estimated cumulative reward for that action. Intuitively, mixing in the uniform distribution is done to make sure that the algorithm tries out all K actions and gets good estimates of the rewards for each. Otherwise, the algorithm might miss a good action because the initial rewards it observes for this action are low and large rewards that occur later are not observed because the action is not selected.

For the drawn action i_t, **Exp3** sets the estimated reward $\hat{x}_{i_t}(t)$ to $x_{i_t}(t)/p_{i_t}(t)$. Dividing the actual gain by the probability that the action was chosen compensates the reward of actions that are unlikely to be chosen. This choice of estimated rewards guarantees that their expectations are equal to the actual rewards for each action; that is, $\mathbf{E}[\hat{x}_j(t) \mid i_1, \ldots, i_{t-1}] = x_j(t)$, where the expectation is taken with respect to

Algorithm Exp3
Parameters: Real $\gamma \in (0,1]$.
Initialization: $w_i(1) = 1$ for $i = 1, \dots, K$.

For each $t = 1, 2, \dots$

1. Set
$$p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \frac{\gamma}{K} \qquad i = 1, \dots, K.$$
2. Draw i_t randomly accordingly to the probabilities $p_1(t), \dots, p_K(t)$.
3. Receive reward $x_{i_t}(t) \in [0,1]$.
4. For $j = 1, \dots, K$ set
$$\hat{x}_j(t) = \begin{cases} x_j(t)/p_j(t) & \text{if } j = i_t, \\ 0 & \text{otherwise,} \end{cases}$$
$$w_j(t+1) = w_j(t) \exp\left(\gamma \hat{x}_j(t)/K\right) .$$

FIG. 1. *Pseudocode of algorithm* **Exp3** *for the weak regret.*

the random choice of i_t at trial t given the choices $i_1, \dots, i_{t-1}$ in the previous $t-1$ trials.

We now give the first main theorem of this paper, which bounds the expected weak regret of algorithm **Exp3**.

THEOREM 3.1. *For any $K > 0$ and for any $\gamma \in (0,1]$,*

$$G_{\max} - \mathbf{E}[G_{\mathbf{Exp3}}] \le (e-1)\gamma G_{\max} + \frac{K \ln K}{\gamma}$$

holds for any assignment of rewards and for any $T > 0$.

To understand this theorem, it is helpful to consider a simpler bound which can be obtained by an appropriate choice of the parameter γ.

COROLLARY 3.2. *For any $T > 0$, assume that $g \ge G_{\max}$ and that algorithm* **Exp3** *is run with input parameter*

$$\gamma = \min\left\{1, \sqrt{\frac{K \ln K}{(e-1)g}}\right\}.$$

Then

$$G_{\max} - \mathbf{E}[G_{\mathbf{Exp3}}] \le 2\sqrt{e-1}\sqrt{gK \ln K} \le 2.63\sqrt{gK \ln K}$$

holds for any assignment of rewards.

Proof. If $g \le (K \ln K)/(e-1)$, then the bound is trivial since the expected regret cannot be more than g. Otherwise, by Theorem 3.1, the expected regret is at most

$$(e-1)\gamma G_{\max} + \frac{K \ln K}{\gamma} \le 2\sqrt{e-1}\sqrt{gK \ln K},$$

as desired. □

To apply Corollary 3.2, it is necessary that an upper bound g on $G_{\max}(T)$ be available for tuning γ. For example, if the time horizon T is known, then, since no action can have payoff greater than 1 on any trial, we can use $g = T$ as an upper bound. In section 4, we give a technique that does not require prior knowledge of such an upper bound, yielding a result which uniformly holds over T.

If the rewards $x_i(t)$ are in the range $[a, b]$, $a < b$, then **Exp3** can be used after the rewards have been translated and rescaled to the range $[0, 1]$. Applying Corollary 3.2 with $g = T$ gives the bound $(b-a)2\sqrt{e-1}\sqrt{TK \ln K}$ on the regret. For instance, this is applicable to a standard loss model where the "rewards" fall in the range $[-1, 0]$.

Proof of Theorem 3.1. The theorem is clearly true for $\gamma = 1$, so assume $0 < \gamma < 1$. Here (and also throughout the paper without explicit mention) we use the following simple facts, which are immediately derived from the definitions:

$$\hat{x}_i(t) \le 1/p_i(t) \le K/\gamma, \tag{2}$$

$$\sum_{i=1}^{K} p_i(t)\hat{x}_i(t) = p_{i_t}(t)\frac{x_{i_t}(t)}{p_{i_t}(t)} = x_{i_t}(t), \tag{3}$$

$$\sum_{i=1}^{K} p_i(t)\hat{x}_i(t)^2 = p_{i_t}(t)\frac{x_{i_t}(t)}{p_{i_t}(t)}\hat{x}_{i_t}(t) \le \hat{x}_{i_t}(t) = \sum_{i=1}^{K} \hat{x}_i(t). \tag{4}$$

Let $W_t = w_1(t) + \cdots + w_K(t)$. For all sequences $i_1, \ldots, i_T$ of actions drawn by **Exp3**,

$$\begin{aligned}
\frac{W_{t+1}}{W_t} &= \sum_{i=1}^{K} \frac{w_i(t+1)}{W_t} \\
&= \sum_{i=1}^{K} \frac{w_i(t)}{W_t} \exp\left(\frac{\gamma}{K}\hat{x}_i(t)\right) \\
&= \sum_{i=1}^{K} \frac{p_i(t) - \frac{\gamma}{K}}{1-\gamma} \exp\left(\frac{\gamma}{K}\hat{x}_i(t)\right) && (5) \\
&\le \sum_{i=1}^{K} \frac{p_i(t) - \frac{\gamma}{K}}{1-\gamma} \left[1 + \frac{\gamma}{K}\hat{x}_i(t) + (e-2)\left(\frac{\gamma}{K}\hat{x}_i(t)\right)^2\right] && (6) \\
&\le 1 + \frac{\frac{\gamma}{K}}{1-\gamma}\sum_{i=1}^{K} p_i(t)\hat{x}_i(t) + \frac{(e-2)(\frac{\gamma}{K})^2}{1-\gamma}\sum_{i=1}^{K} p_i(t)\hat{x}_i(t)^2 && (7) \\
&\le 1 + \frac{\frac{\gamma}{K}}{1-\gamma}x_{i_t}(t) + \frac{(e-2)(\frac{\gamma}{K})^2}{1-\gamma}\sum_{i=1}^{K} \hat{x}_i(t). && (8)
\end{aligned}$$

Equation (5) uses the definition of $p_i(t)$ in Figure 1. Equation (6) uses the fact that $e^x \le 1 + x + (e-2)x^2$ for $x \le 1$; the expression in the preceding line is at most 1 by (2). Equation (8) uses (3) and (4). Taking logarithms and using $1 + x \le e^x$ gives

$$\ln \frac{W_{t+1}}{W_t} \le \frac{\frac{\gamma}{K}}{1-\gamma}x_{i_t}(t) + \frac{(e-2)(\frac{\gamma}{K})^2}{1-\gamma}\sum_{i=1}^{K} \hat{x}_i(t).$$

Summing over t we then get

$$\ln \frac{W_{T+1}}{W_1} \le \frac{\frac{\gamma}{K}}{1-\gamma}G_{\mathbf{Exp3}} + \frac{(e-2)(\frac{\gamma}{K})^2}{1-\gamma}\sum_{t=1}^{T}\sum_{i=1}^{K} \hat{x}_i(t)\ . \tag{9}$$

For any action j,

$$\ln \frac{W_{T+1}}{W_1} \geq \ln \frac{w_j(T+1)}{W_1} = \frac{\gamma}{K} \sum_{t=1}^{T} \hat{x}_j(t) - \ln K.$$

Combining this with (9), we get

$$G_{\mathbf{Exp3}} \geq (1-\gamma) \sum_{t=1}^{T} \hat{x}_j(t) - \frac{K \ln K}{\gamma} - (e-2) \frac{\gamma}{K} \sum_{t=1}^{T} \sum_{i=1}^{K} \hat{x}_i(t) \ . \tag{10}$$

We next take the expectation of both sides of (10) with respect to the distribution of $\langle i_1, \ldots, i_T \rangle$. For the expected value of each $\hat{x}_i(t)$, we have

$$\mathbf{E}[\hat{x}_i(t) \mid i_1, \ldots, i_{t-1}] = \mathbf{E}\left[p_i(t) \cdot \frac{x_i(t)}{p_i(t)} + (1 - p_i(t)) \cdot 0 \right] = x_i(t) \ . \tag{11}$$

Combining (10) and (11), we find that

$$\mathbf{E}[G_{\mathbf{Exp3}}] \geq (1-\gamma) \sum_{t=1}^{T} x_j(t) - \frac{K \ln K}{\gamma} - (e-2) \frac{\gamma}{K} \sum_{t=1}^{T} \sum_{i=1}^{K} x_i(t) \ .$$

Since j was chosen arbitrarily and

$$\sum_{t=1}^{T} \sum_{i=1}^{K} x_i(t) \leq K \, G_{\max},$$

we obtain the inequality in the statement of the theorem. □

Additional notation. As our other player algorithms will be variants of **Exp3**, we find it convenient to define some further notation based on the quantities used in the analysis of **Exp3**.

For each $1 \leq i \leq K$ and for each $t \geq 1$, define

$$G_i(t+1) \stackrel{\text{def}}{=} \sum_{s=1}^{t} x_i(s),$$

$$\hat{G}_i(t+1) \stackrel{\text{def}}{=} \sum_{s=1}^{t} \hat{x}_i(s),$$

$$\hat{G}_{\max}(t+1) \stackrel{\text{def}}{=} \max_{1 \leq i \leq K} \hat{G}_i(t+1).$$

4. Bounds on the weak regret that uniformly hold over time. In section 3, we showed that **Exp3** yields an expected regret of $O(\sqrt{Kg \ln K})$ whenever an upper bound g on the return $G_{\max}$ of the best action is known in advance. A bound of $O(\sqrt{KT \ln K})$, which holds uniformly over T, could be easily proven via the "guessing techniques" which will be used to prove Corollaries 8.4 and 8.5 in section 8. In this section, instead, we describe an algorithm, called **Exp3.1**, whose expected weak regret is $O(\sqrt{KG_{\max} \ln K})$ uniformly over T. As $G_{\max} = G_{\max}(T) \leq T$, this bound is never worse than $O(\sqrt{KT \ln K})$ and is substantially better whenever the return of the best arm is small compared to T.

Algorithm Exp3.1
Initialization: Let $t = 1$, and $\hat{G}_i(1) = 0$ for $i = 1, \ldots, K$.

Repeat for $r = 0, 1, 2, \ldots$

1. Let $g_r = (K \ln K)/(e-1)\, 4^r$.
2. Restart **Exp3** choosing $\gamma_r = \min\left\{1, \sqrt{\frac{K \ln K}{(e-1)g_r}}\right\}$.
3. **While** $\max_i \hat{G}_i(t) \le g_r - K/\gamma_r$ **do**:
   (a) Let i_t be the random action chosen by **Exp3** and $x_{i_t}(t)$ the corresponding reward.
   (b) $\hat{G}_i(t+1) = \hat{G}_i(t) + \hat{x}_i(t)$ for $i = 1, \ldots, K$.
   (c) $t := t + 1$.

FIG. 2. *Pseudocode of algorithm* **Exp3.1** *to control the weak regret uniformly over time.*

Our algorithm **Exp3.1**, described in Figure 2, proceeds in *epochs*, where each epoch consists of a sequence of trials. We use $r = 0, 1, 2, \ldots$ to index the epochs. On epoch r, the algorithm "guesses" a bound g_r for the return of the best action. It then uses this guess to tune the parameter γ of **Exp3**, restarting **Exp3** at the beginning of each epoch. As usual, we use t to denote the current time step.[2] **Exp3.1** maintains an estimate $\hat{G}_i(t+1)$ of the return of each action i. Since $\mathbf{E}[\hat{x}_i(t)] = x_i(t)$, this estimate will be unbiased in the sense that $\mathbf{E}[\hat{G}_i(t+1)] = G_i(t+1)$ for all i and t. Using these estimates, the algorithm detects (approximately) when the actual gain of some action has advanced beyond g_r. When this happens, the algorithm goes on to the next epoch, restarting **Exp3** with a larger bound on the maximal gain.

The performance of the algorithm is characterized by the following theorem which is the main result of this section.

THEOREM 4.1. *For any* $K > 0$,

$$\begin{aligned} G_{\max} - \mathbf{E}[G_{\mathbf{Exp3.1}}] &\le 8\sqrt{e-1}\sqrt{G_{\max} K \ln K} + 8(e-1)K + 2K \ln K \\ &\le 10.5\, \sqrt{G_{\max} K \ln K} + 13.8\, K + 2K \ln K \end{aligned}$$

holds for any assignment of rewards and for any $T > 0$.

The proof of the theorem is divided into two lemmas. The first bounds the regret suffered on each epoch, and the second bounds the total number of epochs.

Fix T arbitrarily and define the following random variables: Let R be the total number of epochs (i.e., the final value of r). Let S_r and T_r be the first and last time steps completed on epoch r (where, for convenience, we define $T_R = T$). Thus, epoch r consists of trials $S_r, S_r + 1, \ldots, T_r$. Note that, in degenerate cases, some epochs may be empty in which case $S_r = T_r + 1$. Let $\hat{G}_{\max} = \hat{G}_{\max}(T+1)$.

LEMMA 4.2. *For any action* j *and for every epoch* r,

$$\sum_{t=S_r}^{T_r} x_{i_t}(t) \ge \sum_{t=S_r}^{T_r} \hat{x}_j(t) - 2\sqrt{e-1}\sqrt{g_r K \ln K}\ .$$

[2] Note that, in general, this t may differ from the "local variable" t used by **Exp3** which we now regard as a subroutine. Throughout this section, we will only use t to refer to the total number of trials as in Figure 2.

Proof. If $S_r > T_r$ (so that no trials occur on epoch r), then the lemma holds trivially since both summations will be equal to zero. Assume then that $S_r \le T_r$. Let $g = g_r$ and $\gamma = \gamma_r$. We use (10) from the proof of Theorem 3.1:

$$\sum_{t=S_r}^{T_r} x_{i_t}(t) \ge \sum_{t=S_r}^{T_r} \hat{x}_j(t) - \gamma \sum_{t=1}^{T_r} \hat{x}_j(t) - \frac{K \ln K}{\gamma} - (e-2)\frac{\gamma}{K} \sum_{t=S_r}^{T_r} \sum_{i=1}^{K} \hat{x}_i(t) \ .$$

From the definition of the termination condition we know that $\hat{G}_i(T_r) \le g - K/\gamma$. Using (2), we get $\hat{x}_i(t) \le K/\gamma$. This implies that $\hat{G}_i(T_r + 1) \le g$ for all i. Thus,

$$\sum_{t=S_r}^{T_r} x_{i_t}(t) \ge \sum_{t=S_r}^{T_r} \hat{x}_j(t) - g\left(\gamma + \gamma(e-2)\right) - \frac{K \ln K}{\gamma} \ .$$

By our choice for γ, we get the statement of the lemma. □

The next lemma gives an implicit upper bound on the number of epochs R. Let $c = (K \ln K)/(e-1)$.

LEMMA 4.3. *The number of epochs R satisfies*

$$2^{R-1} \le \frac{K}{c} + \sqrt{\frac{\hat{G}_{\max}}{c}} + \frac{1}{2} \ .$$

Proof. If $R = 0$, then the bound holds trivially. So assume $R \ge 1$. Let $z = 2^{R-1}$. Because epoch $R-1$ was completed, by the termination condition,

$$\hat{G}_{\max} \ge \hat{G}_{\max}(T_{R-1} + 1) > g_{R-1} - \frac{K}{\gamma_{R-1}} = c\, 4^{R-1} - K\, 2^{R-1} = cz^2 - Kz \ . \tag{12}$$

Suppose the claim of the lemma is false. Then $z > K/c + \sqrt{\hat{G}_{\max}/c}$. Since the function $cx^2 - Kx$ is increasing for $x > K/(2c)$, this implies that

$$cz^2 - Kz > c\left(\frac{K}{c} + \sqrt{\frac{\hat{G}_{\max}}{c}}\right)^2 - K\left(\frac{K}{c} + \sqrt{\frac{\hat{G}_{\max}}{c}}\right) = K\sqrt{\frac{\hat{G}_{\max}}{c}} + \hat{G}_{\max} \ ,$$

contradicting (12). □

Proof of Theorem 4.1. Using the lemmas, we have that

$$\begin{aligned}
G_{\mathbf{Exp3.1}} = \sum_{t=1}^{T} x_{i_t}(t) &= \sum_{r=0}^{R} \sum_{t=S_r}^{T_r} x_{i_t}(t) \\
&\ge \max_j \sum_{r=0}^{R} \left(\sum_{t=S_r}^{T_r} \hat{x}_j(t) - 2\sqrt{e-1}\sqrt{g_r K \ln K} \right) \\
&= \max_j \hat{G}_j(T+1) - 2K \ln K \sum_{r=0}^{R} 2^r \\
&= \hat{G}_{\max} - 2K \ln K (2^{R+1} - 1) \\
&\ge \hat{G}_{\max} + 2K \ln K - 8K \ln K \left(\frac{K}{c} + \sqrt{\frac{\hat{G}_{\max}}{c}} + \frac{1}{2} \right) \\
&= \hat{G}_{\max} - 2K \ln K - 8(e-1)K - 8\sqrt{e-1}\sqrt{\hat{G}_{\max} K \ln K} \ . \qquad (13)
\end{aligned}$$

Here, we used Lemma 4.2 for the first inequality and Lemma 4.3 for the second inequality. The other steps follow from definitions and simple algebra.

Let $f(x) = x - a\sqrt{x} - b$ for $x \geq 0$, where $a = 8\sqrt{e-1}\sqrt{K \ln K}$ and $b = 2K \ln K + 8(e-1)K$. Taking expectations of both sides of (13) gives

$$\mathbf{E}[G_{\mathbf{Exp3.1}}] \geq \mathbf{E}[f(\hat{G}_{\max})] \ . \tag{14}$$

Since the second derivative of f is positive for $x > 0$, f is convex so that, by Jensen's inequality,

$$\mathbf{E}[f(\hat{G}_{\max})] \geq f(\mathbf{E}[\hat{G}_{\max}]) \ . \tag{15}$$

Note that

$$\mathbf{E}[\hat{G}_{\max}] = \mathbf{E}\left[\max_j \hat{G}_j(T+1)\right] \geq \max_j \mathbf{E}[\hat{G}_j(T+1)] = \max_j \sum_{t=1}^{T} x_j(t) = G_{\max} \ .$$

The function f is increasing if and only if $x > a^2/4$. Therefore, if $G_{\max} > a^2/4$, then $f(\mathbf{E}[\hat{G}_{\max}]) \geq f(G_{\max})$. Combined with (14) and (15), this gives that $\mathbf{E}[G_{\mathbf{Exp3.1}}] \geq f(G_{\max})$, which is equivalent to the statement of the theorem. On the other hand, if $G_{\max} \leq a^2/4$, then, because f is nonincreasing on $[0, a^2/4]$,

$$f(G_{\max}) \leq f(0) = -b \leq 0 \leq \mathbf{E}[G_{\mathbf{Exp3.1}}],$$

so the theorem trivially follows in this case as well. □

5. Lower bounds on the weak regret. In this section, we state a lower bound on the expected weak regret of any player. More precisely, for any choice of the time horizon T we show that there exists a strategy for assigning the rewards to the actions such that the expected weak regret of any player algorithm is $\Omega(\sqrt{KT})$. Observe that this does not match the upper bound for our algorithms **Exp3** and **Exp3.1** (see Corollary 3.2 and Theorem 4.1); it is an open problem to close this gap.

Our lower bound is proven using the classical (statistical) bandit model with a crucial difference: the reward distribution depends on the number K of actions and on the time horizon T. This dependence is the reason why our lower bound does not contradict the upper bounds of the form $O(\ln T)$ for the classical bandit model [14]. There, the distribution over the rewards is fixed as $T \to \infty$.

Note that our lower bound has a considerably stronger dependence on the number K of action than the lower bound $\Theta(\sqrt{T \ln K})$, which could have been directly proven from the results in [3, 6]. Specifically, our lower bound implies that no upper bound is possible of the form $O(T^{\alpha}(\ln K)^{\beta})$, where $0 \leq \alpha < 1$, $\beta > 0$.

THEOREM 5.1. *For any number of actions $K \geq 2$ and for any time horizon T, there exists a distribution over the assignment of rewards such that the expected weak regret of any algorithm (where the expectation is taken with respect to both the randomization over rewards and the algorithm's internal randomization) is at least*

$$\frac{1}{20} \min\{\sqrt{KT}, T\}.$$

The proof is given in Appendix A.

The lower bound implies, of course, that for any algorithm there is a particular choice of rewards that will cause the expected weak regret (where the expectation is now with respect to the algorithm's internal randomization only) to be larger than this value.

Algorithm Exp3.P
Parameters: Reals $\alpha > 0$ and $\gamma \in (0, 1]$.
Initialization: For $i = 1, \ldots, K$

$$w_i(1) = \exp\left(\frac{\alpha\gamma}{3}\sqrt{\frac{T}{K}}\right).$$

For each $t = 1, 2, \ldots, T$

1. For $i = 1, \ldots, K$ set

$$p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \frac{\gamma}{K}.$$

2. Choose i_t randomly according to the distribution $p_1(t), \ldots, p_K(t)$.
3. Receive reward $x_{i_t}(t) \in [0, 1]$.
4. For $j = 1, \ldots, K$ set

$$\hat{x}_j(t) = \begin{cases} x_j(t)/p_j(t) & \text{if } j = i_t, \\ 0 & \text{otherwise,} \end{cases}$$

$$w_j(t+1) = w_j(t)\exp\left(\frac{\gamma}{3K}\left(\hat{x}_j(t) + \frac{\alpha}{p_j(t)\sqrt{KT}}\right)\right).$$

FIG. 3. *Pseudocode of algorithm* **Exp3.P** *achieving small weak regret with high probability.*

6. Bounds on the weak regret that hold with probability 1. In section 4 we showed that the *expected* weak regret of algorithm **Exp3.1** is $O(\sqrt{KT \ln K})$. In this section we show that a modification of **Exp3** achieves a weak regret of $O(\sqrt{KT \ln(KT/\delta)})$ with probability at least $1 - \delta$, for any fixed δ and uniformly over T. From this, a bound on the weak regret that holds with probability 1 easily follows.

The modification of **Exp3** is necessary since the variance of the regret achieved by this algorithm is large—so large that an interesting high probability bound may not hold. The large variance of the regret comes from the large variance of the estimates $\hat{x}_i(t)$ for the payoffs $x_i(t)$. In fact, the variance of $\hat{x}_i(t)$ can be close to $1/p_i(t)$, which, for γ in our range of interest, is (ignoring the dependence of K) of magnitude $\sqrt{T}$. Summing over trials, the variance of the return of **Exp3** is about $T^{3/2}$, so that the regret might be as large as $T^{3/4}$.

To control the variance we modify algorithm **Exp3** so that it uses estimates which are based on upper confidence bounds instead of estimates with the correct expectation. The modified algorithm **Exp3.P** is given in Figure 3. Let

$$\hat{\sigma}_i(t+1) \stackrel{\text{def}}{=} \sqrt{KT} + \sum_{s=1}^{t} \frac{1}{p_i(t)\sqrt{KT}}.$$

Whereas algorithm **Exp3** directly uses the estimates $\hat{G}_i(t)$ when choosing i_t at random, algorithm **Exp3.P** uses the upper confidence bounds $\hat{G}_i(t) + \alpha\hat{\sigma}_i(t)$. The next

lemma shows that, for appropriate α, these are indeed upper confidence bounds. Fix some time horizon T. In what follows, we will use $\hat{\sigma}_i$ to denote $\hat{\sigma}_i(T+1)$ and $\hat{G}_i$ to denote $\hat{G}_i(T+1)$.

LEMMA 6.1. *If* $2\sqrt{\ln(KT/\delta)} \le \alpha \le 2\sqrt{KT}$, *then*

$$\mathbf{P}\left\{\exists i : \hat{G}_i + \alpha\hat{\sigma}_i < G_i\right\} \le \delta.$$

Proof. Fix some i and set

$$s_t \stackrel{\text{def}}{=} \frac{\alpha}{2\hat{\sigma}_i(t+1)}.$$

Since $\alpha \le 2\sqrt{KT}$ and $\hat{\sigma}_i(t+1) \ge \sqrt{KT}$, we have $s_t \le 1$. Now

$$\begin{aligned}
&\mathbf{P}\left\{\hat{G}_i + \alpha\hat{\sigma}_i < G_i\right\} \\
&= \mathbf{P}\left\{\sum_{t=1}^{T}(x_i(t) - \hat{x}_i(t)) - \frac{\alpha}{2}\hat{\sigma}_i > \frac{\alpha}{2}\hat{\sigma}_i\right\} \\
&\le \mathbf{P}\left\{s_T\sum_{t=1}^{T}\left(x_i(t) - \hat{x}_i(t) - \frac{\alpha}{2p_i(t)\sqrt{KT}}\right) > \frac{\alpha^2}{4}\right\} \qquad (16) \\
&= \mathbf{P}\left\{\exp\left(s_T\sum_{t=1}^{T}\left(x_i(t) - \hat{x}_i(t) - \frac{\alpha}{2p_i(t)\sqrt{KT}}\right)\right) > \exp\left(\frac{\alpha^2}{4}\right)\right\} \\
&\le e^{-\alpha^2/4}\mathbf{E}\left[\exp\left(s_T\sum_{t=1}^{T}\left(x_i(t) - \hat{x}_i(t) - \frac{\alpha}{2p_i(t)\sqrt{KT}}\right)\right)\right], \qquad (17)
\end{aligned}$$

where in step (16) we multiplied both sides by s_T and used $\hat{\sigma}_i \ge \sum_{t=1}^{T} 1/(p_i(t)\sqrt{KT})$, while in step (17) we used Markov's inequality. For $t = 1, \ldots, T$ set

$$Z_t \stackrel{\text{def}}{=} \exp\left(s_t\sum_{\tau=1}^{t}\left(x_i(\tau) - \hat{x}_i(\tau) - \frac{\alpha}{2p_i(\tau)\sqrt{KT}}\right)\right).$$

Then, for $t = 2, \ldots, T$

$$Z_t = \exp\left(s_t\left(x_i(t) - \hat{x}_i(t) - \frac{\alpha}{2p_i(t)\sqrt{KT}}\right)\right)\cdot(Z_{t-1})^{\frac{s_t}{s_{t-1}}}.$$

Denote by $\mathbf{E}_t[Z_t] = \mathbf{E}[Z_t \mid i_1, \ldots, i_{t-1}]$ the expectation of Z_t with respect to the random choice in trial t and conditioned on the past $t-1$ trials. Note that when the past $t-1$ trials are fixed the only random quantities in Z_t are the $\hat{x}_i(t)$'s. Note also that $x_i(t) - \hat{x}_i(t) \le 1$ and that

$$\begin{aligned}
\mathbf{E}_t\left[(x_i(t) - \hat{x}_i(t))^2\right] &= \mathbf{E}_t\left[\hat{x}_i(t)^2\right] - x_i(t)^2 \\
&\le \mathbf{E}_t\left[\hat{x}_i(t)^2\right] \\
&= \frac{x_i(t)^2}{p_i(t)} \le \frac{1}{p_i(t)}. \qquad (18)
\end{aligned}$$

Hence, for each $t = 2, \ldots, T$

$$\mathbf{E}_t\left[Z_t\right] \le \mathbf{E}_t\left[\exp s_t\left(x_i(t) - \hat{x}_i(t) - \frac{s_t}{p_i(t)}\right)\right]\left(Z_{t-1}\right)^{\frac{s_t}{s_{t-1}}} \tag{19}$$

$$\le \mathbf{E}_t\left[1 + s_t(x_i(t) - \hat{x}_i(t)) + s_t^2(x_i(t) - \hat{x}_i(t))^2\right]\exp\left(-\frac{s_t^2}{p_i(t)}\right)\left(Z_{t-1}\right)^{\frac{s_t}{s_{t-1}}} \tag{20}$$

$$\le \left(1 + s_t^2/p_i(t)\right)\exp\left(-\frac{s_t^2}{p_i(t)}\right)\left(Z_{t-1}\right)^{\frac{s_t}{s_{t-1}}} \tag{21}$$

$$\le \left(Z_{t-1}\right)^{\frac{s_t}{s_{t-1}}} \tag{22}$$

$$\le 1 + Z_{t-1}. \tag{23}$$

Equation (19) uses

$$\frac{\alpha}{2p_i(t)\sqrt{KT}} \ge \frac{\alpha}{2p_i(t)\hat{\sigma}_i(t+1)} = \frac{s_t}{p_i(t)}$$

since $\hat{\sigma}_i(t+1) \ge \sqrt{KT}$. Equation (20) uses $e^a \le 1 + a + a^2$ for $a \le 1$. Equation (21) uses $\mathbf{E}_t\left[\hat{x}_i(t)\right] = x_i(t)$. Equation (22) uses $1 + x \le e^x$ for any real x. Equation (23) uses $s_t \le s_{t-1}$ and $z^u \le 1 + z$ for any $z > 0$ and $u \in [0, 1]$. Observing that $\mathbf{E}\left[Z_1\right] \le 1$, we get by induction that $\mathbf{E}[Z_T] \le T$, and the lemma follows by our choice of α. □

The next lemma shows that the return achieved by algorithm **Exp3**.**P** is close to its upper confidence bounds. Let

$$\hat{U} \stackrel{\text{def}}{=} \max_{1 \le i \le K}\left(\hat{G}_i + \alpha\hat{\sigma}_i\right).$$

LEMMA 6.2. *If* $\alpha \le 2\sqrt{KT}$, *then*

$$G_{\mathbf{Exp3.P}} \ge \left(1 - \frac{5\gamma}{3}\right)\hat{U} - \frac{3}{\gamma}K\ln K - 2\alpha\sqrt{KT} - 2\alpha^2\ .$$

Proof. We proceed as in the analysis of algorithm **Exp3**. Set $\eta = \gamma/(3K)$ and consider any sequence $i_1, \ldots, i_T$ of actions chosen by **Exp3**.**P**. As $\hat{x}_i(t) \le K/\gamma$, $p_i(t) \ge \gamma/K$, and $\alpha \le 2\sqrt{KT}$, we have

$$\eta\hat{x}_i(t) + \frac{\alpha\eta}{p_i(t)\sqrt{KT}} \le 1\ .$$

Therefore,

$$\begin{aligned}
\frac{W_{t+1}}{W_t} &= \sum_{i=1}^{K}\frac{w_i(t+1)}{W_t} \\
&= \sum_{i=1}^{K}\frac{w_i(t)}{W_t}\exp\left(\eta\hat{x}_i(t) + \frac{\alpha\eta}{p_i(t)\sqrt{KT}}\right) \\
&= \sum_{i=1}^{K}\frac{p_i(t) - \gamma/K}{1-\gamma}\exp\left(\eta\hat{x}_i(t) + \frac{\alpha\eta}{p_i(t)\sqrt{KT}}\right) \\
&\le \sum_{i=1}^{K}\frac{p_i(t) - \gamma/K}{1-\gamma}\left[1 + \eta\hat{x}_i(t) + \frac{\alpha\eta}{p_i(t)\sqrt{KT}} + 2\eta^2\hat{x}_i(t)^2 + \frac{2\alpha^2\eta^2}{p_i(t)^2KT}\right]
\end{aligned}$$

$$\leq 1 + \frac{\eta}{1-\gamma} \sum_{i=1}^{K} p_i(t)\hat{x}_i(t) + \frac{\alpha\eta}{1-\gamma} \sum_{i=1}^{K} \frac{1}{\sqrt{KT}}$$
$$+ \frac{2\eta^2}{1-\gamma} \sum_{i=1}^{K} p_i(t)\hat{x}_i(t)^2 + \frac{2\alpha^2\eta^2}{1-\gamma} \sum_{i=1}^{K} \frac{1}{p_i(t)KT}$$
$$\leq 1 + \frac{\eta}{1-\gamma} x_{i_t}(t) + \frac{\alpha\eta}{1-\gamma}\sqrt{\frac{K}{T}} + \frac{2\eta^2}{1-\gamma} \sum_{i=1}^{K} \hat{x}_i(t) + \frac{2\alpha^2\eta}{1-\gamma}\frac{1}{T} \ .$$

The second inequality uses $e^a \leq 1+a+a^2$ for $a \leq 1$, and $(a+b)^2 \leq 2(a^2+b^2)$ for any a, b. The last inequality uses (2), (3), and (4). Taking logarithms, using $\ln(1+x) \leq x$, and summing over $t = 1, \ldots, T$, we get

$$\ln \frac{W_{T+1}}{W_1} \leq \frac{\eta}{1-\gamma} G_{\mathbf{Exp3.P}} + \frac{\alpha\eta}{1-\gamma}\sqrt{KT} + \frac{2\eta^2}{1-\gamma} \sum_{i=1}^{K} \hat{G}_i + \frac{2\alpha^2\eta}{1-\gamma} \ .$$

Since

$$\ln W_1 = \alpha\eta\sqrt{KT} + \ln K$$

and for any j

$$\ln W_{T+1} \geq \ln w_j(T+1) \geq \eta\hat{G}_j + \alpha\eta\hat{\sigma}_j,$$

this implies

$$G_{\mathbf{Exp3.P}} \geq (1-\gamma)\left(\hat{G}_j + \alpha\hat{\sigma}_j\right) - \frac{1}{\eta}\ln K - 2\alpha\sqrt{KT} - 2\eta\sum_{i=1}^{K}\hat{G}_i - 2\alpha^2$$

for any j. Finally, using $\eta = \gamma/(3K)$ and

$$\sum_{i=1}^{K} \hat{G}_i \leq K\hat{U}$$

yields the lemma. □

Combining Lemmas 6.1 and 6.2 gives the main result of this section.

THEOREM 6.3. *For any fixed $T > 0$, for all $K \geq 2$ and for all $\delta > 0$, if*

$$\gamma = \min\left\{\frac{3}{5}, 2\sqrt{\frac{3}{5}\frac{K\ln K}{T}}\right\} \qquad \text{and} \qquad \alpha = 2\sqrt{\ln(KT/\delta)} \ ,$$

then

$$G_{\max} - G_{\mathbf{Exp3.P}} \leq 4\sqrt{KT\ln\frac{KT}{\delta}} + 4\sqrt{\frac{5}{3}KT\ln K} + 8\ln\frac{KT}{\delta}$$

holds for any assignment of rewards with probability at least $1-\delta$.

Proof. We assume without loss of generality that $T \geq (20/3)K\ln K$ and that $\delta \geq KTe^{-KT}$. If either of these conditions do not hold, then the theorem holds trivially. Note that $T \geq (20/3)K\ln K$ ensures $\gamma \leq 3/5$. Note also that $\delta \geq KTe^{-KT}$

Algorithm Exp3.P.1
Parameters: Real $0 < \delta < 1$.
Initialization: For each $r \geq 1$ let $T_r = 2^r$, $\delta_r = \frac{\delta}{(r+1)(r+2)}$, and set

$$r^* = \min\{r \in \mathbb{N} : \delta_r \geq KT_r e^{-KT_r}\}. \tag{24}$$

Repeat for $r = r^*, r^*+1, \ldots$
Run **Exp3.P** for T_r trials choosing α and γ as in Theorem 6.3 with $T = T_r$ and $\delta = \delta_r$.

FIG. 4. *Pseudocode of algorithm* **Exp3.P.1** *(see Theorem* 6.4*).*

implies $\alpha \leq 2\sqrt{KT}$ for our choice of α. So we can apply Lemmas 6.1 and 6.2. By Lemma 6.2 we have

$$G_{\mathbf{Exp3.P}} \geq \left(1 - \frac{5\gamma}{3}\right)\hat{U} - \frac{3}{\gamma}K\ln K - 2\alpha\sqrt{KT} - 2\alpha^2 \ .$$

By Lemma 6.1 we have $\hat{U} \geq G_{\max}$ with probability at least $1 - \delta$. Collecting terms and using $G_{\max} \leq T$ gives the theorem. □

It is not difficult to obtain an algorithm that does not need the time horizon T as input parameter and whose regret is only slightly worse than that proven for the algorithm **Exp3.P** in Theorem 6.3. This new algorithm, called **Exp3.P.1** and shown in Figure 4, simply restarts **Exp3.P** doubling its guess for T each time. The only crucial issue is the choice of the confidence parameter δ and of the minimum length of the runs to ensure that Lemma 6.1 holds for all the runs of **Exp3.P**.

THEOREM 6.4. *Let* r^* *be as in* (24)*. Let* $K \geq 2$*,* $\delta \in (0,1)$*, and* $T \geq 2^{r^*}$*. Let* $c_T = 2\ln(2 + \log_2 T)$*. Then*

$$G_{\max} - G_{\mathbf{Exp3.P.1}} \leq \frac{10}{\sqrt{2}-1}\sqrt{2KT\left(\ln\frac{KT}{\delta} + c_T\right)} + 10(1+\log_2 T)\left(\ln\frac{KT}{\delta} + c_T\right)$$

holds with probability at least $1 - \delta$*.*

Proof. Choose the time horizon T arbitrarily and call *epoch* the sequence of trials between two successive restarts of algorithm **Exp3.P**.

For each $r > r^*$, where r^* is defined in (24), let

$$G_i(r) \stackrel{\text{def}}{=} \sum_{t=2^r+1}^{2^{r+1}} x_i(t) \ , \quad \hat{G}_i(r) \stackrel{\text{def}}{=} \sum_{t=2^r+1}^{2^{r+1}} \hat{x}_i(t) \ , \quad \hat{\sigma}_i(r) \stackrel{\text{def}}{=} \sqrt{KT_r} + \sum_{t=2^r+1}^{2^{r+1}} \frac{1}{p_i(t)\sqrt{KT_r}}$$

and similarly define the quantities $G_i(r^*)$ and $\hat{G}_i(r^*)$ with sums that go from $t = 1$ to $t = 2^{r^*+1}$.

For each $r \geq r^*$, we have $\delta_r \geq KT_r e^{-KT_r}$. Thus we can find numbers α_r such that, by Lemma 6.1,

$$\mathbf{P}\left\{(\exists r \geq r^*)(\exists i) : \hat{G}_i(r) + \alpha_r\hat{\sigma}_i(r) < G_i(r)\right\} \leq \sum_{r=r^*}^{\infty} \mathbf{P}\left\{\exists i : \hat{G}_i(r) + \alpha_r\hat{\sigma}_i(r) < G_i(r)\right\}$$

$$\leq \sum_{r=0}^{\infty} \frac{\delta}{(r+1)(r+2)}$$
$$= \delta \ .$$

We now apply Theorem 6.3 to each epoch. Since $T \geq 2^{r^*}$, there is an $\ell \geq 1$ such that

$$2^{r^*+\ell-1} \leq T = \sum_{r=0}^{\ell-1} 2^{r^*+r} < 2^{r^*+\ell} \ .$$

With probability at least $1-\delta$ over the random draw of **Exp3.P.1**'s actions $i_1, \ldots, i_T$,

$$\begin{aligned}
&G_{\max} - G_{\mathbf{Exp3.P.1}} \\
&\leq \sum_{r=0}^{\ell-1} 10 \left[\sqrt{K T_{r^*+r} \ln \frac{K T_{r^*+r}}{\delta_{r^*+r}}} + \ln \frac{K T_{r^*+r}}{\delta_{r^*+r}} \right] \\
&\leq 10 \left[\sqrt{K \ln \frac{K T_{r^*+\ell-1}}{\delta_{r^*+\ell-1}}} \sum_{r=0}^{\ell-1} \sqrt{T_{r^*+r}} + \ell \ln \frac{K T_{r^*+\ell-1}}{\delta_{r^*+\ell-1}} \right] \\
&\leq 10 \left[\sqrt{K \ln \frac{K T_{r^*+\ell-1}}{\delta_{r^*+\ell-1}}} \left(\frac{2^{(r^*+\ell)/2}}{\sqrt{2}-1} \right) + \ell \ln \frac{K T_{r^*+\ell-1}}{\delta_{r^*+\ell-1}} \right] \\
&\leq \frac{10}{\sqrt{2}-1} \sqrt{2KT \left(\ln \frac{KT}{\delta} + c_T \right)} + 10(1+\log_2 T) \left(\ln \frac{KT}{\delta} + c_T \right),
\end{aligned}$$

where $c_T = 2\ln(2+\log_2 T)$. □

From the above theorem we get, as a simple corollary, a statement about the almost sure convergence of the return of algorithm **Exp3.P**. The rate of convergence is almost optimal, as one can see from our lower bound in section 5.

COROLLARY 6.5. *For any $K \geq 2$ and for any function $f : \mathbb{R} \to \mathbb{R}$ with $\lim_{T\to\infty} f(T) = \infty$,*

$$\lim_{T\to\infty} \frac{G_{\max} - G_{\mathbf{Exp3.P.1}}}{\sqrt{T(\ln T) f(T)}} = 0$$

holds for any assignment of rewards with probability 1.

Proof. Let $\delta = 1/T^2$. Then, by Theorem 6.4, there exists a constant C such that for all T large enough

$$G_{\max} - G_{\mathbf{Exp3.P.1}} \leq C\sqrt{KT \ln T}$$

with probability at least $1 - 1/T^2$. This implies that

$$\mathbf{P}\left\{ \frac{G_{\max} - G_{\mathbf{Exp3.P.1}}}{\sqrt{(T \ln T) f(T)}} > C\sqrt{\frac{K}{f(T)}} \right\} \leq \frac{1}{T^2},$$

and the theorem follows from the Borel–Cantelli lemma. □

7. The regret against the best strategy from a pool. Consider a setting where the player has preliminarily fixed a set of strategies that could be used for choosing actions. These strategies might select different actions at different iterations. The strategies can be computations performed by the player or they can be

external advice given to the player by "experts." We will use the more general term "expert" (borrowed from Cesa-Bianchi et al. [3]) because we place no restrictions on the generation of the advice. The player's goal in this case is to combine the advice of the experts in such a way that its return is close to that of the best expert.

Formally, an expert i is an infinite sequence $\boldsymbol{\xi}^i(1), \boldsymbol{\xi}^i(2), \ldots \in [0,1]^K$ of probability vectors, where the jth component $\xi^i_j(t)$ of $\boldsymbol{\xi}^i(t)$ represents the recommended probability of playing action j at time t. An *adversarial bandit problem with N experts* is thus specified by both an assignment of rewards to actions and by an assignment of probability vectors to each expert. We assume that the player, prior to choosing an action at time t, is provided with the set $\boldsymbol{\xi}^1(t), \ldots, \boldsymbol{\xi}^N(t) \in [0,1]^K$. (As a special case, the distribution can be concentrated on a single action, which represents a deterministic recommendation.) If the vector of rewards at time t is $\boldsymbol{x}(t)$, then the expected reward for expert i, with respect to the chosen probability vector $\boldsymbol{\xi}^i(t)$, is simply $\boldsymbol{\xi}^i(t) \cdot \boldsymbol{x}(t)$. In analogy of $G_{\max}$, we define

$$\tilde{G}_{\max} \stackrel{\text{def}}{=} \max_{1 \le i \le N} \sum_{t=1}^{T} \boldsymbol{\xi}^i(t) \cdot \boldsymbol{x}(t)$$

measuring the expected return of the best strategy. Then the *regret for the best strategy* at time horizon T, defined by $\tilde{G}_{\max}(T) - G_A(T)$, measures the difference between the return of the best expert and player's A return up to time T.

We could at this point view each expert as a "meta-action" in a higher-level bandit problem with payoff vector defined at trial t as $(\boldsymbol{\xi}^1(t) \cdot \boldsymbol{x}(t), \ldots, \boldsymbol{\xi}^N(t) \cdot \boldsymbol{x}(t))$. We could then immediately apply Corollary 3.2 to obtain a bound of $O(\sqrt{gN \log N})$ on the player's regret relative to the best expert (where g is an upper bound on $\tilde{G}_{\max}$). However, this bound is quite weak if the player is combining many experts (i.e., if N is very large). We show below that the algorithm **Exp3** from section 3 can be modified yielding a regret term of the form $O(\sqrt{gK \log N})$. This bound is very reasonable when the number of actions is small, but the number of experts is quite large (even exponential).

Our algorithm **Exp4** is shown in Figure 5, and is only a slightly modified version of **Exp3**. (**Exp4** stands for "exponential-weight algorithm for exploration and exploitation using expert advice.") Let us define $\boldsymbol{y}(t) \in [0,1]^N$ to be the vector with components corresponding to the gains of the experts: $y_i(t) = \boldsymbol{\xi}^i(t) \cdot \boldsymbol{x}(t)$.

The simplest possible expert is one which always assigns uniform weight to all actions so that $\xi_j(t) = 1/K$ on each round t. We call this the *uniform expert*. To prove our results, we need to assume that the uniform expert is included in the family of experts.[3] Clearly, the uniform expert can always be added to any given family of experts at the very small expense of increasing N by one.

THEOREM 7.1. *For any $K, T > 0$, for any $\gamma \in (0,1]$, and for any family of experts which includes the uniform expert,*

$$\tilde{G}_{\max} - \mathbf{E}[G_{\mathbf{Exp4}}] \le (e-1)\gamma \tilde{G}_{\max} + \frac{K \ln N}{\gamma}$$

holds for any assignment of rewards.

[3] In fact, we can use a slightly weaker sufficient condition, namely, that the uniform expert is included in the convex hull of the family of experts, i.e., that there exists nonnegative numbers $\alpha_1, \ldots, \alpha_N$ with $\sum_{j=1}^N \alpha_j = 1$ such that, for all t and all i, $\sum_{j=1}^N \alpha_j \xi^j_i(t) = 1/K$.

Algorithm Exp4
Parameters: Real $\gamma \in (0,1]$.
Initialization: $w_i(1) = 1$ for $i = 1, \ldots, N$.

For each $t = 1, 2, \ldots$

1. Get advice vectors $\boldsymbol{\xi}^1(t), \ldots, \boldsymbol{\xi}^N(t)$.
2. Set $W_t = \sum_{i=1}^N w_i(t)$ and for $j = 1, \ldots, K$ set
$$p_j(t) = (1-\gamma) \sum_{i=1}^N \frac{w_i(t)\xi_j^i(t)}{W_t} + \frac{\gamma}{K} \ .$$
3. Draw action i_t randomly according to the probabilities $p_1(t), \ldots, p_K(t)$.
4. Receive reward $x_{i_t}(t) \in [0,1]$.
5. For $j = 1, \ldots, K$ set
$$\hat{x}_j(t) = \begin{cases} x_j(t)/p_j(t) & \text{if } j = i_t, \\ 0 & \text{otherwise.} \end{cases}$$
6. For $i = 1, \ldots, N$ set
$$\hat{y}_i(t) = \boldsymbol{\xi}^i(t) \cdot \hat{\boldsymbol{x}}(t),$$
$$w_i(t+1) = w_i(t) \exp\left(\gamma \hat{y}_i(t)/K\right) \ .$$

FIG. 5. *Pseudocode of algorithm* **Exp4** *for using expert advice.*

Proof. We prove this theorem along the lines of the proof of Theorem 3.1. Let $q_i(t) = w_i(t)/W_t$. Then

$$\begin{aligned} \frac{W_{t+1}}{W_t} &= \sum_{i=1}^N \frac{w_i(t+1)}{W_t} \\ &= \sum_{i=1}^N q_i(t) \exp\left(\frac{\gamma}{K}\hat{y}_i(t)\right) \\ &\le \sum_{i=1}^N q_i(t) \left[1 + \frac{\gamma}{K}\hat{y}_i(t) + (e-2)\left(\frac{\gamma}{K}\hat{y}_i(t)\right)^2\right] \\ &= 1 + \left(\frac{\gamma}{K}\right) \sum_{i=1}^N q_i(t)\hat{y}_i(t) + (e-2)\left(\frac{\gamma}{K}\right)^2 \sum_{i=1}^N q_i(t)\hat{y}_i(t)^2 \ . \end{aligned}$$

Taking logarithms and summing over t we get

$$\ln \frac{W_{T+1}}{W_1} \le \left(\frac{\gamma}{K}\right) \sum_{t=1}^T \sum_{i=1}^N q_i(t)\hat{y}_i(t) + (e-2)\left(\frac{\gamma}{K}\right)^2 \sum_{t=1}^T \sum_{i=1}^N q_i(t)\hat{y}_i(t)^2 \ .$$

Since, for any expert k,

$$\ln \frac{W_{T+1}}{W_1} \geq \ln \frac{w_k(T+1)}{W_1} = \frac{\gamma}{K} \sum_{t=1}^{T} \hat{y}_k(t) - \ln N,$$

we get

$$\sum_{t=1}^{T} \sum_{i=1}^{N} q_i(t) \hat{y}_i(t) \geq \sum_{t=1}^{T} \hat{y}_k(t) - \frac{K \ln N}{\gamma} - (e-2) \frac{\gamma}{K} \sum_{t=1}^{T} \sum_{i=1}^{N} q_i(t) \hat{y}_i(t)^2 \ .$$

Note that

$$\begin{aligned} \sum_{i=1}^{N} q_i(t) \hat{y}_i(t) &= \sum_{i=1}^{N} q_i(t) \left(\sum_{j=1}^{K} \xi_j^i(t) \hat{x}_j(t) \right) \\ &= \sum_{j=1}^{K} \left(\sum_{i=1}^{N} q_i(t) \xi_j^i(t) \right) \hat{x}_j(t) \\ &= \sum_{j=1}^{K} \left(\frac{p_j(t) - \frac{\gamma}{K}}{1-\gamma} \right) \hat{x}_j(t) \leq \frac{x_j(t)}{1-\gamma} \ . \end{aligned}$$

Also,

$$\begin{aligned} \sum_{i=1}^{N} q_i(t) \hat{y}_i(t)^2 &= \sum_{i=1}^{N} q_i(t) (\xi_{i_t}^i(t) \hat{x}_{i_t}(t))^2 \\ &\leq \hat{x}_{i_t}(t)^2 \frac{p_{i_t}(t)}{1-\gamma} \\ &\leq \frac{\hat{x}_{i_t}(t)}{1-\gamma} \ . \end{aligned}$$

Therefore, for all experts k,

$$G_{\mathbf{Exp4}} = \sum_{t=1}^{T} \hat{x}_{i_t}(t) \geq (1-\gamma) \sum_{t=1}^{T} \hat{y}_k(t) - \frac{K \ln N}{\gamma} - (e-2) \frac{\gamma}{K} \sum_{t=1}^{T} \sum_{j=1}^{K} \hat{x}_j(t) \ .$$

We now take expectations of both sides of this inequality. Note that

$$\mathbf{E}[\hat{y}_k(t)] = \mathbf{E} \left[\sum_{j=1}^{K} \xi_j^k(t) \hat{x}_j(t) \right] = \sum_{j=1}^{K} \xi_j^k(t) x_j(t) = y_k(t) \ .$$

Further,

$$\frac{1}{K} \mathbf{E} \left[\sum_{t=1}^{T} \sum_{j=1}^{K} \hat{x}_j(t) \right] = \sum_{t=1}^{T} \frac{1}{K} \sum_{j=1}^{K} x_j(t) \leq \max_{1 \leq i \leq N} \sum_{t=1}^{T} y_i(t) = \tilde{G}_{\max}$$

since we have assumed that the uniform expert is included in the family of experts. Combining these facts immediately implies the statement of the theorem. □

Algorithm Exp3.S
Parameters: Reals $\gamma \in (0,1]$ and $\alpha > 0$.
Initialization: $w_i(1) = 1$ for $i = 1, \ldots, K$.

For each $t = 1, 2, \ldots$

1. Set

$$p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \frac{\gamma}{K}, \qquad i = 1, \ldots, K.$$

2. Draw i_t according to the probabilities $p_1(t), \ldots, p_K(t)$.
3. Receive reward $x_{i_t}(t) \in [0,1]$.
4. For $j = 1, \ldots, K$ set

$$\hat{x}_j(t) = \begin{cases} x_j(t)/p_j(t) & \text{if } j = i_t, \\ 0 & \text{otherwise,} \end{cases}$$

$$w_j(t+1) = w_j(t)\, \exp\left(\gamma \hat{x}_j(t)/K\right) + \frac{e\alpha}{K} \sum_{i=1}^{K} w_i(t)\ .$$

FIG. 6. *Pseudocode of algorithm* **Exp3.S** *to control the expected regret.*

8. The regret against arbitrary strategies. In this section we present a variant of algorithm **Exp3** and prove a bound on its expected regret for any sequence $(j_1, \ldots, j_T)$ of actions. To prove this result, we rank all sequences of actions according to their "hardness." The *hardness* of a sequence $(j_1, \ldots, j_T)$ is defined by

$$\mathrm{H}(j_1, \ldots, j_T) \stackrel{\text{def}}{=} 1 + \left|\{1 \le \ell < T : j_\ell \ne j_{\ell+1}\}\right|.$$

So, $\mathrm{H}(1, \ldots, 1) = 1$ and $\mathrm{H}(1, 1, 3, 2, 2) = 3$. The bound on the regret which we will prove grows with the hardness of the sequence for which we are measuring the regret. In particular, we will show that the player algorithm **Exp3.S** described in Figure 6 has an expected regret of $O(\mathrm{H}(j^T)\sqrt{KT\ln(KT)})$ for any sequence $j^T = (j_1, \ldots, j_T)$ of actions. On the other hand, if the regret is measured for any sequence j^T of actions of hardness $\mathrm{H}(j^T) \le S$, then the expected regret of **Exp3.S** (with parameters tuned to this S) reduces to $O(\sqrt{SKT\ln(KT)})$. In what follows, we will use G_{j^T} to denote the return $x_{j_1}(1) + \cdots + x_{j_T}(T)$ of a sequence $j^T = (j_1, \ldots, j_T)$ of actions.

THEOREM 8.1. *For any $K > 0$, for any $\gamma \in (0,1]$, and for any $\alpha > 0$,*

$$G_{j^T} - \mathbf{E}\left[G_{\mathbf{Exp3.S}}\right] \le \frac{K(\mathrm{H}(j^T)\ln(K/\alpha) + e\alpha T)}{\gamma} + (e-1)\gamma T$$

holds for any assignment of rewards, for any $T > 0$, and for any sequence $j^T = (j_1, \ldots, j_T)$ of actions.

COROLLARY 8.2. *Assume that algorithm* **Exp3.S** *is run with input parameters $\alpha = 1/T$ and*

$$\gamma = \min\left\{1, \sqrt{\frac{K\ln(KT)}{T}}\right\}\ .$$

Then

$$G_{j^T} - \mathbf{E}\left[G_{\mathbf{Exp3.S}}\right] \leq \mathrm{H}(j^T)\sqrt{KT\ln(KT)} + 2e\sqrt{\frac{KT}{\ln(KT)}}$$

holds for any sequence $j^T = (j_1, \ldots, j_T)$ *of actions.*

Note that the statement of Corollary 8.2 can be equivalently written as

$$\mathbf{E}\left[G_{\mathbf{Exp3.S}}\right] \geq \max_{j^T}\left(G_{j^T} - \mathrm{H}(j^T)\sqrt{KT\ln(KT)}\right) - 2e\sqrt{\frac{KT}{\ln(KT)}},$$

revealing that algorithm **Exp3.S** is able to automatically trade-off between the return G_{j^T} of a sequence j^T and its hardness $\mathrm{H}(j^T)$.

COROLLARY 8.3. *Assume that algorithm* **Exp3.S** *is run with input parameters* $\alpha = 1/T$ *and*

$$\gamma = \min\left\{1, \sqrt{\frac{K(S\ln(KT) + e)}{(e-1)T}}\right\}.$$

Then

$$G_{j^T} - \mathbf{E}\left[G_{\mathbf{Exp3.S}}\right] \leq 2\sqrt{e-1}\sqrt{KT\left(S\ln(KT) + e\right)}$$

holds for any sequence $j^T = (j_1, \ldots, j_T)$ *of actions such that* $\mathrm{H}(j^T) \leq S$.

Proof of Theorem 8.1. Fix any sequence $j^T = (j_1, \ldots, j_T)$ of actions. With a technique that closely follows the proof of Theorem 3.1, we can prove that for all sequences $i_1, \ldots, i_T$ of actions drawn by **Exp3.S**,

$$\frac{W_{t+1}}{W_t} \leq 1 + \frac{\gamma/K}{1-\gamma}x_{i_t}(t) + \frac{(e-2)(\gamma/K)^2}{1-\gamma}\sum_{i=1}^{K}\hat{x}_i(t) + e\alpha, \tag{25}$$

where, as usual, $W_t = w_1(t) + \cdots + w_K(t)$. Now let $S = \mathrm{H}(j^T)$ and partition $(1, \ldots, T)$ in *segments*

$$[T_1, \ldots, T_2), [T_2, \ldots, T_3), \ldots, [T_S, \ldots, T_{S+1}),$$

where $T_1 = 1$, $T_{S+1} = T + 1$, and $j_{T_s} = j_{T_s+1} = \cdots = j_{T_{s+1}-1}$ for each segment $s = 1, \ldots, S$. Fix an arbitrary segment $[T_s, T_{s+1})$ and let $\Delta_s = T_{s+1} - T_s$. Furthermore, let

$$G_{\mathbf{Exp3.S}}(s) \stackrel{\text{def}}{=} \sum_{t=T_s}^{T_{s+1}-1} x_{i_t}(t)\ .$$

Taking logarithms on both sides of (25) and summing over $t = T_s, \ldots, T_{s+1} - 1$, we get

$$\ln\frac{W_{T_{s+1}}}{W_{T_s}} \leq \frac{\gamma/K}{1-\gamma}G_{\mathbf{Exp3.S}}(s) + \frac{(e-2)(\gamma/K)^2}{1-\gamma}\sum_{t=T_s}^{T_{s+1}-1}\sum_{i=1}^{K}\hat{x}_i(t) + e\alpha\Delta_s\ . \tag{26}$$

Now let j be the action such that $j_{T_s} = \cdots = j_{T_{s+1}-1} = j$. Since

$$\begin{aligned} w_j(T_{s+1}) &\geq w_j(T_s+1)\exp\left(\frac{\gamma}{K}\sum_{t=T_s+1}^{T_{s+1}-1}\hat{x}_j(t)\right) \\ &\geq \frac{e\alpha}{K}W_{T_s}\exp\left(\frac{\gamma}{K}\sum_{t=T_s+1}^{T_{s+1}-1}\hat{x}_j(t)\right) \\ &\geq \frac{\alpha}{K}W_{T_s}\exp\left(\frac{\gamma}{K}\sum_{t=T_s}^{T_{s+1}-1}\hat{x}_j(t)\right), \end{aligned}$$

where the last step uses $\gamma\hat{x}_j(t)/K \leq 1$, we have

$$\ln\frac{W_{T_{s+1}}}{W_{T_s}} \geq \ln\frac{w_j(T_{s+1})}{W_{T_s}} \geq \ln\left(\frac{\alpha}{K}\right) + \frac{\gamma}{K}\sum_{t=T_s}^{T_{s+1}-1}\hat{x}_j(t)\ . \tag{27}$$

Piecing together (26) and (27) we get

$$G_{\mathbf{Exp3.S}}(s) \geq (1-\gamma)\sum_{t=T_s}^{T_{s+1}-1}\hat{x}_j(t) - \frac{K\ln(\frac{K}{\alpha})}{\gamma} - (e-2)\frac{\gamma}{K}\sum_{t=T_s}^{T_{s+1}-1}\sum_{i=1}^{K}\hat{x}_i(t) - \frac{e\alpha K\Delta_s}{\gamma}\ .$$

Summing over all segments $s = 1, \ldots, S$, taking expectation with respect to the random choices of algorithm **Exp3.S**, and using

$$G_{(j_1,\ldots,j_T)} \leq T \qquad \text{and} \qquad \sum_{t=1}^{T}\sum_{i=1}^{K}x_i(t) \leq KT$$

yields the inequality in the statement of the theorem. □

If the time horizon T is not known, we can apply techniques similar to those applied for proving Theorem 6.4 in section 6. More specifically, we introduce a new algorithm, **Exp3.S.1**, that runs **Exp3.S** as a subroutine. Suppose that at each new run (or epoch) $r = 0, 1, \ldots$, **Exp3.S** is started with its parameters set as prescribed in Corollary 8.2, where T is set to $T_r = 2^r$, and then stopped after T_r iterations. Clearly, for any fixed sequence $j^T = (j_1, \ldots, j_T)$ of actions, the number of segments (see proof of Theorem 8.1 for a definition of segment) within each epoch r is at most $\mathrm{H}(j^T)$. Hence the expected regret of **Exp3.S.1** for epoch r is certainly not more than

$$(\mathrm{H}(j^T) + 2e)\sqrt{KT_r\ln(KT_r)}\ .$$

Let ℓ be such that $2^\ell \leq T < 2^{\ell+1}$. Then the last epoch is $\ell \leq \log_2 T$ and the overall regret (over the $\ell+1$ epochs) is at most

$$(\mathrm{H}(j^T) + 2e)\sum_{r=0}^{\ell}\sqrt{KT_r\ln(KT_r)} \leq (\mathrm{H}(j^T) + 2e)\sqrt{K\ln(KT_\ell)}\sum_{r=0}^{\ell}\sqrt{T_r}\ .$$

Finishing up the calculations proves the following.

COROLLARY 8.4.

$$G_{j^T} - \mathbf{E}\left[G_{\mathbf{Exp3.S.1}}\right] \leq \frac{\mathrm{H}(j^T) + 2e}{\sqrt{2}-1}\sqrt{2KT\ln(KT)}$$

for any $T > 0$ and for any sequence $j^T = (j_1, \dots, j_T)$ of actions.

On the other hand, if **Exp3.S.1** runs **Exp3.S** with parameters set as prescribed in Corollary 8.3, with a reasoning similar to the one above we conclude the following.

COROLLARY 8.5.

$$G_{j^T} - \mathbf{E}\left[G_{\mathbf{Exp3.S.1}}\right] \leq \frac{2\sqrt{e-1}}{\sqrt{2}-1}\sqrt{2KT(S\ln(KT)+e)}$$

for any $T > 0$ and for any sequence $j^T = (j_1, \dots, j_T)$ of actions such that $\mathrm{H}(j^T) \leq S$.

9. Applications to game theory. The adversarial bandit problem can be easily related to the problem of playing repeated games. For $N > 1$ integer, an N-person finite game is defined by N finite sets $S_1, \dots, S_N$ of pure strategies, one set for each player, and by N functions $u_1, \dots, u_N$, where function $u_i : S_1 \times \cdots \times S_N \to \mathbb{R}$ is player's i payoff function. Note the each player's payoff depends both on the pure strategy chosen by the player and on the pure strategies chosen by the other players. Let $S = S_1 \times \cdots \times S_N$, and let $S_{-i} = S_1 \times \cdots \times S_{i-1} \times S_{i+1} \times \cdots \times S_N$. We use $\boldsymbol{s}$ and $\boldsymbol{s}_{-i}$ to denote typical members of, respectively, S and S_{-i}. Given $\boldsymbol{s} \in S$, we will often write $(j, \boldsymbol{s}_{-i})$ to denote $(s_1, \dots, s_{i-1}, j, s_{i+1}, \dots, s_N)$, where $j \in S_i$. Suppose that the game is repeatedly played through time. Assume for now that each player knows all payoff functions and, after each repetition (or round) t, also knows the vector $\boldsymbol{s}(t) = (s_1(t), \dots, s_N(t))$ of pure strategies chosen by the players. Hence, the pure strategy $s_i(t)$ chosen by player i at round t may depend on what player i and the other players chose in the past rounds. The *average regret* of player i for the pure strategy j after T rounds is defined by

$$R_i^{(j)}(T) = \frac{1}{T}\sum_{t=1}^{T}\left[u_i(j, \boldsymbol{s}_{-i}(t)) - u_i(\boldsymbol{s}(t))\right].$$

This is how much player i lost on average for not playing the pure strategy j on all rounds, given that all the other players kept their choices fixed.

A desirable property for a player is Hannan-consistency [8], defined as follows. Player i is *Hannan-consistent* if

$$\limsup_{T\to\infty} \max_{j\in S_i} R_i^{(j)}(T) = 0 \qquad \text{with probability 1.}$$

The existence and properties of Hannan-consistent players have been first investigated by Hannan [10] and Blackwell [2] and later by many others (see [5] for a nice survey).

Hannan-consistency can be also studied in the so-called unknown game setup, where it is further assumed that (1) each player knows neither the total number of players nor the payoff function of any player (including itself), (2) after each round each player sees its own payoffs but it sees neither the choices of the other players nor the resulting payoffs. This setup was previously studied by Baños [1], Megiddo [16], and by Hart and Mas-Colell [11, 12].

We can apply the results of section 6 to prove that a player using algorithm **Exp3.P.1** as mixed strategy is Hannan-consistent in the unknown game setup whenever the payoffs obtained by the player belong to a known bounded real interval. To do that, we must first extend our results to the case when the assignment of rewards can be chosen adaptively. More precisely, we can view the payoff $x_{i_t}(t)$, received by the gambler at trial t of the bandit problem, as the payoff $u_i(i_t, \boldsymbol{s}_{-i}(t))$ received by player i at the tth round of the game. However, unlike our adversarial bandit

framework where all the rewards were assigned to each arm at the beginning, here the payoff $u_i(i_t, \boldsymbol{s}_{-i}(t))$ depends on the (possibly randomized) choices of all players, which, in turn, are functions of their realized payoffs. In our bandit terminology, this corresponds to assuming that the vector $(x_1(t), \ldots, x_K(t))$ of rewards for each trial t is chosen by an adversary who knows the gambler's strategy and the outcome of the gambler's random draws up to time $t-1$. We leave to the interested reader the easy but lengthy task of checking that *all* of our results (including those of section 6) hold under this additional assumption.

Using Theorem 6.4 we then get the following.

THEOREM 9.1. *If player i has $K \geq 2$ pure strategies and plays in the unknown game setup (with payoffs in $[0,1]$) using the mixed strategy* **Exp3.P.1**, *then*

$$\max_{j \in S_i} R_i^{(j)}(T) \leq \frac{10}{\sqrt{2}-1} \sqrt{\frac{2K}{T}\left(\ln \frac{KT}{\delta} + c_T\right)} + \frac{10(1+\log_2 T)}{T}\left(\ln \frac{KT}{\delta} + c_T\right),$$

where $c_T = 2\ln(2+\log_2 T)$, holds with probability at least $1-\delta$, for all $0 < \delta < 1$ and for all $T \geq (\ln(K/\delta))^{1/(K-1)}$.

The constraint on T in the statement of Theorem 9.1 is derived from the condition $T \geq 2^{r^*}$ in Theorem 6.4. Note that, according to Theorem 5.1, the rate of convergence is optimal both in T and K up to logarithmic factors.

Theorem 9.1, along with Corollary 6.5, immediately implies the result below.

COROLLARY 9.2. *Player's strategy* **Exp3.P.1** *is Hannan-consistent in the unknown game setup.*

As pointed out in [5], Hannan-consistency has an interesting consequence for repeated zero-sum games. These games are defined by an $n \times m$ matrix $\mathbf{M}$. On each round t, the *row player* chooses a row i of the matrix. At the same time, the *column player* chooses a column j. The row player then gains the quantity $\mathbf{M}_{ij}$, while the column player loses the same quantity. In repeated play, the row player's goal is to maximize its expected total gain over a sequence of plays, while the column player's goal is to minimize its expected total loss.

Suppose in some round the row player chooses its next move i randomly according to a probability distribution on rows represented by a (column) vector $\boldsymbol{p} \in [0,1]^n$, and the column player similarly chooses according to a probability vector $\boldsymbol{q} \in [0,1]^m$. Then the expected payoff is $\boldsymbol{p}^T \mathbf{M} \boldsymbol{q}$. Von Neumann's minimax theorem states that

$$\max_{\boldsymbol{p}} \min_{\boldsymbol{q}} \boldsymbol{p}^T \mathbf{M} \boldsymbol{q} = \min_{\boldsymbol{q}} \max_{\boldsymbol{p}} \boldsymbol{p}^T \mathbf{M} \boldsymbol{q} \ ,$$

where maximum and minimum are taken over the (compact) set of all distribution vectors $\boldsymbol{p}$ and $\boldsymbol{q}$. The quantity v defined by the above equation is called the *value* of the zero-sum game with matrix $\mathbf{M}$. In words, this says that there exists a mixed (randomized) strategy $\overline{\mathbf{p}}$ for the row player that guarantees expected payoff at least v, regardless of the column player's action. Moreover, this payoff is optimal in the sense that the column player can choose a mixed strategy whose expected payoff is at most v, regardless of the row player's action. Thus, if the row player knows the matrix $\mathbf{M}$, it can compute a strategy (for instance, using linear programming) that is certain to bring an expected optimal payoff not smaller than v on each round.

Suppose now that the game $\mathbf{M}$ is entirely unknown to the row player. To be precise, assume the row player knows only the number of rows of the matrix and a bound on the magnitude of the entries of $\mathbf{M}$. Then, using the results of section 4, we

can show that the row player can play in such a manner that its payoff per round will rapidly converge to the optimal maximin payoff v.

THEOREM 9.3. *Let* $\mathbf{M}$ *be an unknown game matrix in* $[a,b]^{n\times m}$ *with value* v. *Suppose the row player, knowing only* a, b, *and* n, *uses the mixed strategy* **Exp3.1**. *Then the row player's expected payoff per round is at least*

$$v - 8(b-a)\left(\sqrt{e-1}\sqrt{\frac{n\ln n}{T}} - 8(e-1)\frac{n}{T} - 2\frac{n\ln n}{T}\right).$$

Proof. We assume that $[a,b]=[0,1]$; the extension to the general case is straightforward. By Theorem 4.1, we have

$$\begin{aligned}\mathbf{E}\left[\sum_{t=1}^{T}\mathbf{M}_{i_t j_t}\right] &= \mathbf{E}\left[\sum_{t=1}^{T} x_{i_t}(t)\right] \\ &\geq \max_i \mathbf{E}\left[\sum_{t=1}^{T} x_i(t)\right] - 8\sqrt{e-1}\sqrt{Tn\ln n} - 8(e-1)n - 2n\ln n\ .\end{aligned}$$

Let $\overline{\mathbf{p}}$ be a maxmin strategy for the row player such that

$$v = \max_{\boldsymbol{p}}\min_{\boldsymbol{q}} \boldsymbol{p}^T\mathbf{M}\boldsymbol{q} = \min_{\boldsymbol{q}} \overline{\mathbf{p}}^T\mathbf{M}\boldsymbol{q}\ ,$$

and let $\boldsymbol{q}(t)$ be a distribution vector whose j_tth component is 1. Then

$$\max_i \mathbf{E}\left[\sum_{t=1}^{T} x_i(t)\right] \geq \sum_{i=1}^{n}\overline{p}_i\mathbf{E}\left[\sum_{t=1}^{T} x_i(t)\right] = \mathbf{E}\left[\sum_{t=1}^{T}\overline{\mathbf{p}}\cdot\boldsymbol{x}(t)\right] = \mathbf{E}\left[\sum_{t=1}^{T}\overline{\mathbf{p}}^T\mathbf{M}\boldsymbol{q}(t)\right] \geq vT$$

since $\overline{\mathbf{p}}^T\mathbf{M}\boldsymbol{q}\geq v$ for all $\boldsymbol{q}$.

Thus, the row player's expected payoff is at least

$$vT - 8\sqrt{e-1}\sqrt{Tn\ln n} - 8(e-1)n - 2n\ln n\ .$$

Dividing by T to get the average per-round payoff gives the result. □

Note that the theorem is independent of the number of columns of $\mathbf{M}$ and, with appropriate assumptions, the theorem can be easily generalized to column players with an infinite number of strategies. If the matrix $\mathbf{M}$ is very large and all entries are small, then, even if $\mathbf{M}$ is known to the player, our algorithm may be an efficient alternative to linear programming.

Appendix A. Proof of Theorem 5.1. We construct the random distribution of rewards as follows. First, before play begins, one action I is chosen uniformly at random to be the "good" action. The T rewards $x_I(t)$ associated with the good action are chosen independently at random to be 1 with probability $1/2+\epsilon$ and 0 otherwise for some small, fixed constant $\epsilon\in(0,1/2]$ to be chosen later in the proof. The rewards $x_j(t)$ associated with the other actions $j\neq I$ are chosen independently at random to be 0 or 1 with equal odds. Then the expected reward of the best action is at least $(1/2+\epsilon)T$. The main part of the proof below is a derivation of an upper bound on the expected gain of any algorithm for this distribution of rewards.

We write $\mathbf{P}_*\{\cdot\}$ to denote probability with respect to this random choice of rewards, and we also write $\mathbf{P}_i\{\cdot\}$ to denote probability conditioned on i being the good

action: $\mathbf{P}_i\{\cdot\} = \mathbf{P}_*\{\cdot \mid I = i\}$. Finally, we write $\mathbf{P}_{unif}\{\cdot\}$ to denote probability with respect to a uniformly random choice of rewards for *all* actions (including the good action). Analogous expectation notation $\mathbf{E}_*[\cdot]$, $\mathbf{E}_i[\cdot]$, and $\mathbf{E}_{unif}[\cdot]$ will also be used.

Let A be the player strategy. Let $r_t = x_{i_t}(t)$ be a random variable denoting the reward received at time t, and let $\mathbf{r}^t$ denote the sequence of rewards received up through trial t: $\mathbf{r}^t = \langle r_1, \ldots, r_t \rangle$. For shorthand, $\mathbf{r} = \mathbf{r}^T$ is the entire sequence of rewards.

Any randomized playing strategy is equivalent to an a priori random choice from the set of all deterministic strategies. Thus, because the adversary strategy we have defined is oblivious to the actions of the player, it suffices to prove an upper bound on the expected gain of any *deterministic* strategy (this is not crucial for the proof but simplifies the notation). Therefore, we can formally regard the algorithm A as a fixed function which, at each step t, maps the reward history $\mathbf{r}^{t-1}$ to its next action i_t.

As usual, $G_A = \sum_{t=1}^{T} r_t$ denotes the return of the algorithm, and $G_{\max} = \max_j \sum_{t=1}^{T} x_j(t)$ is the return of the best action.

Let N_i be a random variable denoting the number of times action i is chosen by A. Our first lemma bounds the difference between expectations when measured using $\mathbf{E}_i[\cdot]$ or $\mathbf{E}_{unif}[\cdot]$.

LEMMA A.1. *Let $f : \{0,1\}^T \to [0, M]$ be any function defined on reward sequences $\mathbf{r}$. Then for any action i,*

$$\mathbf{E}_i[f(\mathbf{r})] \leq \mathbf{E}_{unif}[f(\mathbf{r})] + \frac{M}{2}\sqrt{-\mathbf{E}_{unif}[N_i]\ln(1-4\epsilon^2)}.$$

Proof. We apply standard methods that can be found, for instance, in Cover and Thomas [4]. For any distributions $\mathbf{P}$ and $\mathbf{Q}$, let

$$\|\mathbf{P} - \mathbf{Q}\|_1 \doteq \sum_{\mathbf{r} \in \{0,1\}^T} |\mathbf{P}\{\mathbf{r}\} - \mathbf{Q}\{\mathbf{r}\}|$$

be the variational distance, and let

$$\mathrm{KL}(\mathbf{P} \parallel \mathbf{Q}) \doteq \sum_{\mathbf{r} \in \{0,1\}^T} \mathbf{P}\{\mathbf{r}\} \lg\left(\frac{\mathbf{P}\{\mathbf{r}\}}{\mathbf{Q}\{\mathbf{r}\}}\right)$$

be the Kullback–Liebler divergence or relative entropy between the two distributions. (We use lg to denote $\log_2$.) We also use the notation

$$\mathrm{KL}\left(\mathbf{P}\{r_t \mid \mathbf{r}^{t-1}\} \parallel \mathbf{Q}\{r_t \mid \mathbf{r}^{t-1}\}\right) \doteq \sum_{\mathbf{r}^t \in \{0,1\}^t} \mathbf{P}\{\mathbf{r}^t\} \lg\left(\frac{\mathbf{P}\{r_t \mid \mathbf{r}^{t-1}\}}{\mathbf{Q}\{r_t \mid \mathbf{r}^{t-1}\}}\right)$$

for the conditional relative entropy of r_t given $\mathbf{r}^{t-1}$. Finally, for $p, q \in [0,1]$, we use

$$\mathrm{KL}(p \parallel q) \doteq p \lg\left(\frac{p}{q}\right) + (1-p)\lg\left(\frac{1-p}{1-q}\right)$$

as shorthand for the relative entropy between two Bernoulli random variables with parameters p and q.

We have that

$$\begin{aligned}\mathbf{E}_i\left[f(\mathbf{r})\right] - \mathbf{E}_{unif}\left[f(\mathbf{r})\right] &= \sum_{\mathbf{r}} f(\mathbf{r})(\mathbf{P}_i\{\mathbf{r}\} - \mathbf{P}_{unif}\{\mathbf{r}\}) \\ &\leq \sum_{\mathbf{r}:\mathbf{P}_i\{\mathbf{r}\}\geq\mathbf{P}_{unif}\{\mathbf{r}\}} f(\mathbf{r})(\mathbf{P}_i\{\mathbf{r}\} - \mathbf{P}_{unif}\{\mathbf{r}\}) \\ &\leq M \sum_{\mathbf{r}:\mathbf{P}_i\{\mathbf{r}\}\geq\mathbf{P}_{unif}\{\mathbf{r}\}} (\mathbf{P}_i\{\mathbf{r}\} - \mathbf{P}_{unif}\{\mathbf{r}\}) \\ &= \frac{M}{2}\|\mathbf{P}_i - \mathbf{P}_{unif}\|_1. \qquad (28)\end{aligned}$$

Also, Cover and Thomas's Lemma 12.6.1 states that

$$\|\mathbf{P}_{unif} - \mathbf{P}_i\|_1^2 \leq (2\ln 2)\mathrm{KL}\left(\mathbf{P}_{unif} \;\|\; \mathbf{P}_i\right). \qquad (29)$$

The "chain rule for relative entropy" (Cover and Thomas's Theorem 2.5.3) gives that

$$\begin{aligned}\mathrm{KL}\left(\mathbf{P}_{unif} \;\|\; \mathbf{P}_i\right) &= \sum_{t=1}^{T} \mathrm{KL}\left(\mathbf{P}_{unif}\{r_t \mid \mathbf{r}^{t-1}\} \;\|\; \mathbf{P}_i\{r_t \mid \mathbf{r}^{t-1}\}\right) \\ &= \sum_{t=1}^{T} \left(\mathbf{P}_{unif}\{i_t \neq i\}\ \mathrm{KL}\left(\tfrac{1}{2} \;\|\; \tfrac{1}{2}\right) + \mathbf{P}_{unif}\{i_t = i\}\ \mathrm{KL}\left(\tfrac{1}{2} \;\|\; \tfrac{1}{2}+\epsilon\right)\right) \\ &= \sum_{t=1}^{T} \mathbf{P}_{unif}\{i_t = i\}\left(-\tfrac{1}{2}\lg(1-4\epsilon^2)\right) \\ &= \mathbf{E}_{unif}\left[N_i\right]\left(-\tfrac{1}{2}\lg(1-4\epsilon^2)\right). \qquad (30)\end{aligned}$$

The second equality can be seen as follows: Regardless of the past history of rewards $\mathbf{r}^{t-1}$, the conditional probability distribution $\mathbf{P}_{unif}\{r_t \mid \mathbf{r}^{t-1}\}$ on the next reward r_t is uniform on $\{0,1\}$. The conditional distribution $\mathbf{P}_i\{r_t \mid \mathbf{r}^{t-1}\}$ is also easily computed: Given $\mathbf{r}^{t-1}$, the next action i_t is fixed by A. If this action is not the good action i, then the conditional distribution is uniform on $\{0,1\}$; otherwise, if $i_t = i$, then r_t is 1 with probability $1/2+\epsilon$ and 0 otherwise.

The lemma now follows by combining (28), (29), and (30). □

We are now ready to prove the theorem. Specifically, we show the following.

THEOREM A.2. *For any player strategy A, and for the distribution on rewards described above, the expected regret of algorithm A is lower bounded by*

$$\mathbf{E}_*\left[G_{\max} - G_A\right] \geq \epsilon\left(T - \frac{T}{K} - \frac{T}{2}\sqrt{-\frac{T}{K}\ln(1-4\epsilon^2)}\right).$$

Proof. If action i is chosen to be the good action, then clearly the expected payoff at time t is $1/2+\epsilon$ if $i_t = i$ and $1/2$ if $i_t \neq i$:

$$\begin{aligned}\mathbf{E}_i\left[r_t\right] &= \left(\tfrac{1}{2}+\epsilon\right)\mathbf{P}_i\{i_t = i\} + \tfrac{1}{2}\mathbf{P}_i\{i_t \neq i\} \\ &= \tfrac{1}{2} + \epsilon\ \mathbf{P}_i\{i_t = i\}.\end{aligned}$$

Thus, the expected gain of algorithm A is

$$\mathbf{E}_i\left[G_A\right] = \sum_{t=1}^{T} \mathbf{E}_i\left[r_t\right] = \frac{T}{2} + \epsilon\ \mathbf{E}_i\left[N_i\right]. \qquad (31)$$

Next, we apply Lemma A.1 to N_i, which is a function of the reward sequence $\mathbf{r}$ since the actions of player strategy A are determined by the past rewards. Clearly, $N_i \in [0, T]$. Thus,

$$\mathbf{E}_i [N_i] \leq \mathbf{E}_{unif} [N_i] + \frac{T}{2} \sqrt{-\mathbf{E}_{unif} [N_i] \ln(1 - 4\epsilon^2)},$$

and so

$$\begin{aligned} \sum_{i=1}^{K} \mathbf{E}_i [N_i] &\leq \sum_{i=1}^{K} \left(\mathbf{E}_{unif} [N_i] + \frac{T}{2} \sqrt{-\mathbf{E}_{unif} [N_i] \ln(1 - 4\epsilon^2)} \right) \\ &\leq T + \frac{T}{2} \sqrt{-TK \ln(1 - 4\epsilon^2)} \end{aligned}$$

using the fact that $\sum_{i=1}^{K} \mathbf{E}_{unif} [N_i] = T$, which implies that $\sum_{i=1}^{K} \sqrt{\mathbf{E}_{unif} [N_i]} \leq \sqrt{TK}$. Therefore, combining with (31),

$$\mathbf{E}_* [G_A] = \frac{1}{K} \sum_{i=1}^{K} \mathbf{E}_i [G_A] \leq \frac{T}{2} + \epsilon \left(\frac{T}{K} + \frac{T}{2} \sqrt{-\frac{T}{K} \ln(1 - 4\epsilon^2)} \right).$$

The expected gain of the best action is at least the expected gain of the good action, so $\mathbf{E}_* [G_{\max}] \geq T(1/2 + \epsilon)$. Thus, we get that the regret is lower bounded by the bound given in the statement of the theorem. □

For small ϵ, the bound given in Theorem A.2 is of the order

$$\Theta \left(T\epsilon - T\epsilon^2 \sqrt{\frac{T}{K}} \right).$$

Choosing $\epsilon = c\sqrt{K/T}$ for some small constant c gives a lower bound of $\Omega(\sqrt{KT})$. Specifically, the lower bound given in Theorem 5.1 is obtained from Theorem A.2 by choosing $\epsilon = (1/4) \min\{\sqrt{K/T}, 1\}$ and using the inequality $-\ln(1-x) \leq (4 \ln(4/3))x$ for $x \in [0, 1/4]$.

Acknowledgments. We express special thanks to Kurt Hornik for his advice and his patience when listening to our ideas and proofs for an earlier draft of this paper. We thank Yuval Peres and Amir Dembo for their help regarding the analysis of martingales. Last but not least we want to acknowledge the work of the anonymous referees whose very precise remarks helped us to improve the paper.

REFERENCES

[1] A. Baños, *On pseudo-games*, Annals of Mathematical Statistics, 39 (1968), pp. 1932–1945.
[2] D. Blackwell, *Controlled random walks*, in Proceedings of the International Congress of Mathematicians, Vol. III, North–Holland, Amsterdam, 1956, pp. 336–338.
[3] N. Cesa-Bianchi, Y. Freund, D. Haussler, D. P. Helmbold, R. E. Schapire, and M. K. Warmuth, *How to use expert advice*, J. ACM, 44 (1997), pp. 427–485.
[4] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley, New York, 1991.
[5] D. P. Foster and R. Vohra, *Regret in the on-line decision problem*, Games Econom. Behav., 29 (1999), pp. 7–36.
[6] Y. Freund and R. E. Schapire, *A decision-theoretic generalization of on-line learning and an application to boosting*, J. Comput. System Sci., 55 (1997), pp. 119–139.

[7] Y. Freund and R. E. Schapire, *Adaptive game playing using multiplicative weights*, Games Econom. Behav., 29 (1999), pp. 79–103.
[8] D. Fudenberg and D. K. Levine, *Consistency and cautious fictitious play*, J. Econom. Dynam. Control, 19 (1995), pp. 1065–1089.
[9] J. C. Gittins, *Multi-armed Bandit Allocation Indices*, John Wiley, Chichester, UK, 1989.
[10] J. Hannan, *Approximation to Bayes risk in repeated play*, in Contributions to the Theory of Games, Vol. III, M. Dresher, A. W. Tucker, and P. Wolfe, eds., Princeton University Press, Princeton, NJ, 1957, pp. 97–139.
[11] S. Hart and A. Mas-Colell, *A simple procedure leading to correlated equilibrium*, Econometrica, 68 (2000), pp. 1127–1150.
[12] S. Hart and A. Mas-Colell, *A general class of adaptive strategies*, J. Econom. Theory, 98 (2001), pp. 26–54.
[13] T. Ishikida and P. Varaiya, *Multi-armed bandit problem revisited*, J. Optim. Theory Appl., 83 (1994), pp. 113–154.
[14] T. L. Lai and H. Robbins, *Asymptotically efficient adaptive allocation rules*, Adv. in Appl. Math., 6 (1985), pp. 4–22.
[15] N. Littlestone and M. K. Warmuth, *The weighted majority algorithm*, Inform. and Comput., 108 (1994), pp. 212–261.
[16] N. Megiddo, *On repeated games with incomplete information played by non-Bayesian players*, International J. Game Theory, 9 (1980), pp. 157–167.
[17] H. Robbins, *Some aspects of the sequential design of experiments*, Bull. Amer. Math. Soc., 55 (1952), pp. 527–535.
[18] V. G. Vovk, *Aggregating strategies*, in Proceedings of the Third Annual Workshop on Computational Learning Theory, Morgan Kaufmann, San Francisco, CA, 1990, pp. 371–386.

SIAM J. COMPUT.
Vol. 32, No. 1, pp. 78–130

A VIRTUALLY SYNCHRONOUS GROUP MULTICAST ALGORITHM FOR WANS: FORMAL APPROACH*

IDIT KEIDAR† AND ROGER KHAZAN‡

Abstract. This paper presents a formal design for a novel group communication service targeted for wide-area networks (WANs). The service provides virtual synchrony semantics. Such semantics facilitate the design of fault tolerant distributed applications. The presented design is more suitable for WANs than previously suggested ones. In particular, it features the first algorithm to achieve virtual synchrony semantics in a single communication round. The design also employs a scalable WAN-oriented architecture: it effectively decouples the main two components of virtually synchronous group communication—group membership and reliable group multicast. The design is carried out formally and rigorously. This paper includes formal specifications of both safety and liveness properties. The algorithm is formally modeled and assertionally verified.

Key words. group communication, virtual synchrony, reliable multicast, formal modeling

AMS subject classifications. 68M14, 68M15, 68W15, 68Q85

PII. S0097539700378754

1. Introduction.

Group communication services (GCSs) [1, 10] are powerful middleware systems that facilitate the development of fault tolerant distributed applications. These services provide a notion of *group abstraction*, which allows application processes to easily organize themselves into multicast groups. Application processes can communicate with the members of a group by addressing messages to the group. Most GCSs strive to present different members of the same group with mutually consistent perceptions of the communication done in the group. This perception is known as *virtual synchrony* (VS) semantics [12].

Traditionally, GCSs were designed for deployment in local-area networks (LANs). Efficient GCSs that operate in wide-area networks (WANs) is still an open area of research. Designing such GCSs is challenging because in WANs communication is more expensive and connectivity is less stable than in LANs.

In this paper we present a novel algorithm for a GCS targeted for WANs. The service provided by our GCS satisfies a variant of the VS semantics that has been shown to be useful for facilitating the design of distributed applications [16, 10]. Our algorithm for implementing this semantics is more appropriate for WANs than the existing solutions: it requires fewer rounds of communication and is designed for the scalable WAN-oriented architecture of [6, 31]. Our design is carried out at a very high level of formality and rigor, much higher than that of most previous designs of virtually synchronous GCSs. It includes formal and precise specifications, algorithms,

*Received by the editors September 27, 2000; accepted for publication (in revised form) June 25, 2002; published electronically November 19, 2002. This paper is an extended version of A client-server approach to virtually synchronous group multicast: Specifications and algorithms, 20th International Conference on Distributed Computing Systems (ICDCS), 2000, IEEE, pp. 344–355. This work was supported by Air Force Aerospace Research (OSR) grants F49620-00-1-0097 and F49620-00-1-0327, Nippon Telegraph and Telephone (NTT) grant MIT9904-12, and NSF grants CCR-9909114 and EIA-9901592.

http://www.siam.org/journals/sicomp/32-1/37875.html

†Laboratory for Computer Science, Massachusetts Institute of Technology, 200 Technology Square, Cambridge, MA 02139. Current address: Department of Electrical Engineering, Technion - Israel Institute of Technology, Technion City, Haifa, 32000 Israel (idish@ee.technion.ac.il).

‡Laboratory for Computer Science, Massachusetts Institute of Technology, 200 Technology Square, Cambridge, MA 02139 (roger@lcs.mit.edu).

and proofs.

The rest of this section is organized as follows: In section 1.1 we present some basic background on GCSs and VS. Section 1.2 summarizes the contributions made by our work, and section 1.3 gives a roadmap to the rest of the paper.

1.1. Background. Modern distributed applications often involve large groups of geographically distributed processes that interact by sending messages over an asynchronous fault-prone network. Many of these applications maintain a replicated state of some sort. In order for these applications to be correct, the replicas must remain mutually consistent throughout the execution of the application. For example, in an online game, the states of the game maintained by different players must be mutually consistent in order for the game to be meaningful to the players. Designing algorithms that maintain state consistency is difficult however: different application processes may perceive the execution of the application inconsistently because of asynchrony and failures. For example, if Alice, Bob, and Carol are playing an online game, the following asymmetric scenario is possible: Alice and Bob perceive each other as alive and well, but they differ in the way they perceive Carol; one sees Carol as crashed or disconnected, while the other sees her as alive and well. Middleware systems that hide from the application some of the underlying inconsistencies and instead present them with a more consistent picture of the distributed execution facilitate development of distributed applications.

GCSs, such as [3, 5, 44, 12], are examples of such middleware systems. They are particularly useful for building applications that require reliable multipoint to multipoint communication among a group (or groups) of processes. Examples of such applications are data replication (for example, [29, 4, 22, 33, 24]), highly available servers (for example, [8]), and online games. GCSs allow application processes to organize themselves easily into groups and to communicate with all the members of a group by addressing messages to the group. The semantics of this abstraction are such that different members of the group have consistent perceptions of the communication done in the group. The abstraction is typically implemented through the integration of two types of services: membership and reliable multicast.

Membership services maintain information about membership of groups. The membership of a group can change dynamically due to new processes joining and current members departing, failing, or disconnecting. The membership service tracks these changes and reports them to group members. The report given by the membership service to a member is called a *view*. It includes a unique identifier and a list of currently active and mutually connected members. Failures can partition a group into disconnected components of mutually connected members. Membership services strive to form and deliver the same views to all mutually connected members of the group.[1] While this is not always possible, they typically succeed once network connectivity more or less stabilizes (see, for example, [31, 16]).

In addition, GCSs provide reliable multicast services that allow application processes to send messages to the entire membership of a group. GCSs guarantee that message delivery satisfies certain properties. For example, one property can be that messages sent by the same sender are delivered in the order in which they were sent; another property can ensure that all processes receive all messages in the same total order. Different GCSs differ in the specific message delivery properties they provide,

[1] In this paper, we consider *partitionable* membership services, which may deliver concurrent, disjoint views of the same group to disconnected members.

but most of them provide some variant of *VS* semantics. We refer to a GCS providing such semantics as a *virtually synchronous GCS*, and to an algorithm implementing this semantics as a *VS algorithm*.

VS semantics specifies how message deliveries are synchronized with view deliveries. This synchronization is done in a way that simulates a "benign" world in which message delivery is reliable within each view. Many variants of VS have been suggested (for example, [38, 23, 16, 12, 40, 9]). Nearly all of them include a key property, called *virtually synchronous delivery*, which guarantees that *processes that receive the same pair of views from the GCS receive the same sets of messages in between receiving the views.* Henceforth, when we refer to VS, we assume the semantics includes virtually synchronous delivery.

EXAMPLE 1.1. *Assume Alice, Bob, and Carol are playing an online game. Assume they communicate using totally ordered messages and modify their game states when they receive messages. Each of them is initially given a view* $\langle\{Alice, Bob, Carol\}, 1\rangle$*, where* $\{Alice, Bob, Carol\}$ *is a set of members and* 1 *is a view identifier. Then Carol disconnects, and Alice and Bob are given a new view* $\langle\{Alice, Bob\}, 2\rangle$*. The virtually synchronous delivery property guarantees that both Alice and Bob receive the same messages before receiving the new view. In particular, if Bob receives a message from Carol before it receives the new view, then Alice also receives this message before the new view. Therefore, Alice and Bob remain in consistent states and can safely continue playing the game after they receive the new view.*

In general, virtually synchronous GCSs are especially useful for building applications that maintain a replicated state of some sort using a variant of the well-known *state-machine/active replication* approach [34, 41] and [32, Chapter 10]. With such an approach, processes that maintain state replicas are organized into multicast groups. Actions that update the state are sent using a multicast primitive that delivers messages to different processes in the same order. When processes receive these actions, they apply them to their local replicas. VS guarantees that processes that remain connected receive the same messages. This implies that processes that remain connected apply the same sequences of actions to their replicas. Hence, their replicas remain mutually consistent. Examples of GCS applications that use this technique are [2, 4, 29, 42, 24, 8].

Let us consider what is involved in implementing the virtually synchronous delivery property. Imagine that GCS processes are forming a new view because someone has disconnected from their current view. The GCS processes must make sure that they deliver the same messages to their application clients before delivering to them the new view. However, it may be the case that some of these GCS processes received messages that others did not. In the scenario illustrated in Example 1.1, the last messages from Carol may have reached the GCS process of only Bob, and not of Alice; Bob and Alice need to agree on whether or not to deliver these messages. To ensure such agreement, GCS processes invoke a *synchronization* protocol whenever a new view is forming.

Designing correct and efficient algorithms that implement VS is not trivial. Different GCS processes may perceive connectivity changes inconsistently. Since the desired synchronization depends on who the members of the new view are, the algorithm has to tolerate transient inconsistent views and cascading connectivity changes.

In particular, a VS algorithm needs to know which synchronization messages sent by different processes pertain to the same view formation attempt. Existing algorithms, such as [23, 3, 40, 9, 26, 5], identify such synchronization messages by

tagging them with a common identifier. Some initial communication is performed first, before synchronization messages are communicated, in order to agree upon a common identifier and to distribute it to the members of the forming view.

While a view is forming and a synchronization protocol is executing, there may be changes in connectivity that call for views with altogether different memberships. When such situations happen, existing VS algorithms, for example [23, 26, 40, 9, 5], continue executing their current synchronization protocol to termination and then deliver to the application a view that does not reflect the already detected changes in connectivity; we refer to such views as *obsolete* [31]. Afterwards, the algorithm is invoked anew to incorporate the new changes. Obsolete views cause an overhead not just for the GCS but also for applications. Since application processes do not know when the views delivered to them are obsolete, they handle such views just as they do any other view, for example, by running state synchronization protocols [29, 22, 33].

1.2. Our contributions. In this paper, we present a novel design for a virtually synchronous GCS targeted for WANs. We make the following contributions:

1. We present a new algorithm for implementing VS. Our algorithm neither processes nor delivers views with obsolete memberships. Moreover, the synchronization protocol run by our algorithm involves just a single message exchange round among members of the new view. We are not aware of any other algorithm for implementing VS that has these two features.

2. Our design demonstrates how to effectively decouple the algorithm for achieving VS from the algorithm for maintaining membership. As suggested in [6, 31], such effective decoupling is important for providing scalable GCSs in WANs.

We define a membership service interface that allows the VS algorithm to execute in parallel with the membership algorithm. In contrast to previous designs, for example, [40, 11], we allow the membership algorithm to freely change memberships of forming views at any time. Moreover, the interaction between the membership and VS algorithms is only in one direction, from the former to the latter. Our interface was adopted by the Moshe [31] membership algorithm; other existing membership algorithms (for example, [20, 5]) can also be easily extended to provide the required interface and semantics.

3. Our design is carried out much more rigorously and formally than most previous designs of virtually synchronous GCSs. The presented specifications of our GCS and its environment, description of the algorithm, and proof of correctness are all precise and formal. Our project is the first to use formal methods for modeling a virtually synchronous GCS and to provide an assertional proof of its correctness.

Our algorithm has been implemented [43] (in C++) as part of a novel architecture for scalable group communication in WANs using the datagram service of [7] and the Moshe membership algorithm [31].

1.3. Roadmap. The rest of this paper is organized as follows. Section 2 gives an overview of our algorithm and overall design. Section 3 reviews the formal model and notation. In section 4 we present the client-server architecture of our GCS and formally specify the assumptions we make on the membership service and the underlying communication substrate. Section 5 contains precise specifications of the safety and liveness properties satisfied by our GCS. The algorithm is then given in section 6 and is accompanied by informal correctness arguments. Section 7 concludes the paper. A formal correctness proof that the algorithm of section 6 satisfies the specifications of section 5 is given in the appendixes: safety properties are given in Appendix B, and liveness properties are given in Appendix C. Appendix A reviews

the proof techniques used in Appendixes B and C.

2. Design overview. The novelty of our algorithm for achieving VS is concentrated in its synchronization protocol. Recall that this protocol is run among GCS processes in order for those that remain connected to agree upon a common set of messages each of them must deliver before moving into the new view. The protocol depends on a simple yet powerful idea. Instead of using common identifiers to designate which synchronization messages pertain to the same view formation attempt, we use locally generated identifiers. These identifiers are then included as part of the formed views.[2] Once a view formation completes at a GCS process, the process knows which synchronization messages of other members to consider for the view—the messages tagged with the identifiers that are included in the view.

EXAMPLE 2.1. *View* $\langle 8, \{Alice, Bob, Carol\}, [4, 3, 7]\} \rangle$ *has membership* $\{Alice, Bob, Carol\}$*, vector of local identifiers* $[4, 3, 7]$*, and view identifier* 8*. When a GCS process forms this view, it uses the synchronization messages from Alice, Bob, and Carol tagged, respectively, with* 4*,* 3*, and* 7 *to decide on the set of messages it must deliver before delivering this view to its application. Thus, if Alice, Bob, and Carol form the same view, they use the same synchronization messages, and thus agree on which application messages each of them needs to deliver.*

The use of local identifiers eliminates the need to preagree on common identifiers and allows the synchronization protocol to complete in a single message exchange round. It also allows the algorithm to promptly react to connectivity changes without wasting resources on obsolete views. The protocol works correctly even if, because of network instability, GCS processes send multiple synchronization messages during the same synchronization protocol.

2.1. Architecture for WAN. Our design decouples the algorithm for implementing virtually synchronous multicast from the algorithm for maintaining membership. The membership algorithm handles generation of local identifiers and formation of views. The algorithm for implementing virtually synchronous multicast synchronizes views and application messages to implement the VS semantics. In particular, it handles multicast requests submitted by the application, delivers application messages and views back to the application, and runs the synchronization protocol to synchronize processes that transition together into new views. The decoupling involves low-cost, one-directional communication from the membership to the virtually synchronous multicast algorithm. It also allows the synchronization protocol to execute in parallel with the membership algorithm forming views.

Efficient decoupling of membership and virtually synchronous multicast algorithms allows for an architecture in which the membership service is implemented by a small set of dedicated membership servers maintaining the membership information on behalf of a large set of clients. This architecture was proposed in [6, 31] for supporting scalable membership services in WANs. Our work extends this architecture by specifying how it can be used as a base for a virtually synchronous GCS. In particular, we present precise specifications of the interface and semantics that a membership service has to provide in order to be decoupled from the virtually synchronous multicast algorithm.

The interface consists of two types of messages, `start` and `view`, sent from membership servers to the processes executing the virtually synchronous multicast algo-

[2] A similar view structure is suggested in [40] for the purpose of not having concurrent views intersect.

rithm; we call these processes the GCS end-points. A `start` message is sent when a membership server starts forming a new view or adds new members to an already forming view. Each `start` message contains a prospective membership set and an identifier, which is *not* globally agreed upon; that is, different processes can be given different identifiers. A `view` message is sent when a server succeeds in forming a new view. The `view` message contains information that maps GCS end-points to the last start identifiers they were given prior to this view. The servers do not need to hear back from the end-points in order to complete the membership algorithm, and the end-points do not impose any restrictions on the servers' choices of views. These two features are in contrast with previously suggested external membership services, such as, for example, Maestro [11] and the service of [40], in which membership servers are not allowed to add new members once view formation begins and, furthermore, have to synchronize with the processes executing the virtually synchronous multicast algorithm before they can produce new views.

2.2. Algorithm for virtual synchrony. When the membership service starts forming a view, it sends a `start` notification with a prospective membership set and a new local identifier to each end-point `p` executing the virtually synchronous multicast algorithm. Upon receiving this notification, `p` sends a synchronization message tagged with this identifier to the end-points in the prospective membership set. In the synchronization message, `p` specifies its current view and the set of application messages that `p` commits to deliver in its current view before delivering the new view. If an end-point `q` joins the membership while a view formation is in progress, `p` will receive a new `start` notification and will then forward to `q` the same synchronization messages it sent when the view formation started.

When `p` receives a `view` message from its membership server, the local identifiers included in the view tell `p` which synchronization messages to consider—those messages that are tagged with the local identifiers included in the new view. Using the information contained in these messages, `p` computes two things: (a) the set of end-points, called the *transitional set* [16], containing those members of the new view that would transition into the new view together with `p` directly from `p`'s current view; and (b) the set of application messages that `p` must deliver in its current view before transitioning into the new view. End-point `p` computes the transitional set to include every end-point `q` that is a member of both `p`'s current view and the new view, and whose synchronization message was sent in the same view as `p`'s current view. As far as the set of application messages, end-point `p` decides on delivering the maximal set of messages identified by the synchronization messages of the transitional set members. Since the same views formed by different end-points contain the same local identifiers, the end-points use the same synchronization messages to compute the transitional set and the message set. After delivering the decided set of application messages, `p` delivers the new view and the transitional set to its application client. The transitional set tells the client about the members of the new view with whom the client is already synchronized.

Unlike previous algorithms, for example, those in [5, 26, 9, 40], our algorithm allows the membership service to change the membership of a forming view while the synchronization protocol is running; the protocol responds immediately to such membership changes. The following example demonstrates the benefits of this approach.

EXAMPLE 2.2. *Figure* 2.1 *presents a sample execution involving two clients,* `a` *and* `b`. *Vertical arrows represent the time passage at each client, empty circles represent* GCS*-level events, and gray circles represent* MEMB*-level events. In order*

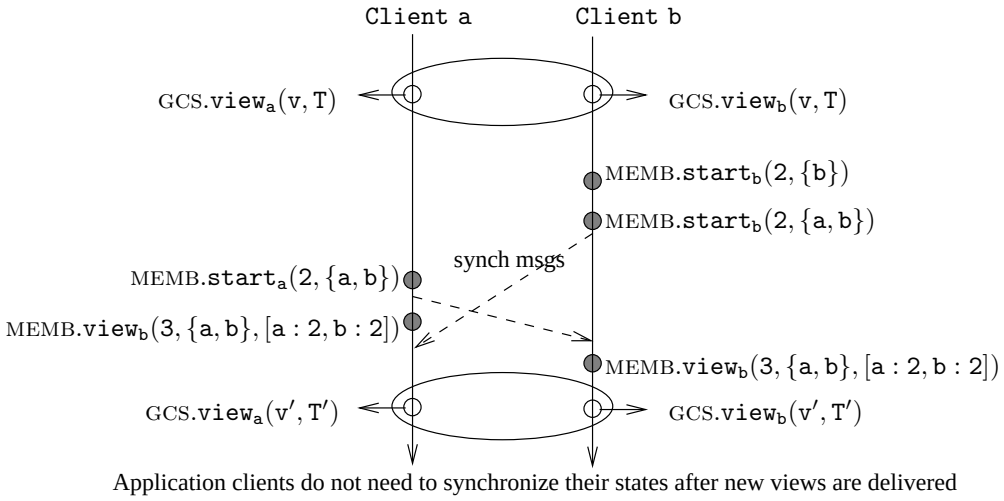


FIG. 2.1. *Handling membership changes while synchronization protocol is running.*

to disambiguate these events, we prefix events generated at the membership server by MEMB *and events generated by the virtually synchronous multicast algorithm by* GCS.

First, both clients receive the same view $\mathtt{v} = \langle 2, \{\mathtt{a}, \mathtt{b}\}, [\mathtt{a} : 1, \mathtt{b} : 1]\rangle$ *from their* GCS *end-points,* $\text{GCS}_\mathtt{a}$ *and* $\text{GCS}_\mathtt{b}$*; the ellipse around these view events highlights that the delivered views are the same. At some point, the* MEMB *service notifies* $\text{GCS}_\mathtt{b}$ *that it is starting to form a view without* a*. While doing so, it detects that* a *is connected to* b *after all, so it changes the membership of the forming view to* $\{\mathtt{a}, \mathtt{b}\}$*.* $\text{GCS}_\mathtt{b}$ *forwards to* $\text{GCS}_\mathtt{a}$ *its latest synchronization message; synchronization messages are denoted by dashed lines.* $\text{GCS}_\mathtt{a}$ *is also notified by* MEMB *of its attempt to form a new view with* b*; this causes* $\text{GCS}_\mathtt{a}$ *to send a synchronization message to* $\text{GCS}_\mathtt{b}$*. When* MEMB *completes its view formation, it delivers the new view* $\mathtt{v}' = \langle 3, \{\mathtt{a}, \mathtt{b}\}, [\mathtt{a} : 2, \mathtt{b} : 2]\rangle$ *to both* GCS *end-points. After the* GCS *end-points receive each others' synchronization messages, they compute their transitional sets to be* $\mathtt{T}' = \{\mathtt{a}, \mathtt{b}\}$*, decide on which application messages they need to deliver, deliver these messages, and then deliver* $\mathtt{v}'$ *and* $\mathtt{T}'$ *to their clients. From* $\mathtt{T}'$*,* a *and* b *can deduce that, due to virtually synchronous delivery, they received the same messages while in* v*, and therefore do not need to synchronize their states.*

Example 2.2 demonstrates two additional advantages of our algorithm: (a) the algorithm does not waste resources on synchronizing end-points in order to deliver views that are known to be obsolete; and (b) the application benefits from not seeing obsolete views, as it has to do fewer state synchronizations (or other view processing activity). Responding promptly to connectivity changes is therefore especially important in WANs, where transient connectivity changes may occur frequently due to variability of message latency and less reliable connectivity. In contrast to our algorithm, algorithms that do not allow new members to be added to the membership of an already forming view (such as [5, 26, 9, 40]) lack these advantages, as illustrated by the following example.

EXAMPLE 2.3. *When executed in the scenario of Example* 2.2*, algorithms that do not allow new members to be added to the membership of an already forming view would deliver an obsolete view* $\mathtt{v}_\mathtt{mid}$ *with membership* $\{\mathtt{b}\}$ *to client* b *and then restart*

the view formation and synchronization protocols anew in order to deliver to a *and* b *a new view with membership* {a, b}. *As part of the synchronization protocol,* a *and* b *would first exchange messages to agree upon a common identifier before actually exchanging synchronization messages. The synchronization protocol would* not *synchronize end-points* a *and* b *because they would be transitioning into the new view from different views,* a *from* v *and* b *from* v_{mid}. *As a result, after the clients get the new view from* GCS, *they would have to run an additional state synchronization protocol.*

2.3. Formal methodology. Our design has been carried out and is presented at a level more formal and rigorous than that of most previous designs of virtually synchronous GCSs. We precisely specify the properties satisfied by our virtually synchronous multicast algorithm, the external membership service, and the underlying communication substrate. We then give a formal description of the virtually synchronous multicast algorithm. The algorithm is accompanied by a careful formal correctness proof. The safety properties are proved by using invariant assertions and simulation mappings; the liveness properties are proved by using invariant assertions and careful operational arguments. We found this level of rigor to be important: in the process of specifying and verifying the algorithm, we uncovered several ambiguities and errors.

Previously, formal approaches were used to specify the semantics of virtually synchronous GCSs and to model and verify their applications, for example, in [15, 22, 18, 33, 27]. Existing algorithms implementing VS are modeled in pseudocode and proven correct operationally. However, due to their size and complexity, such algorithms were not previously modeled using formal methods nor were they assertionally verified.

To manage the complexity of this project we have developed a formal inheritance-based methodology [30] for incrementally constructing specifications, algorithms, and proofs. In addition to making the project tractable, the use of this construct makes clear which parts of the algorithm implement which property. The modularity of this approach facilitates further modifications and alterations of the design. Our project and the inheritance-based construct are both developed in the framework of the I/O automaton formalism (see [37] and [36], Chap. 8).

3. Formal model and notation. In the I/O automaton model ([37] and [36], Chap. 8), a system component is described as a state-machine, called an *I/O automaton*. The transitions of this state-machine are associated with named actions, which are classified as either *input*, *output*, or *internal*. Input and output actions model the component's interaction with other components, while internal actions are externally unobservable.

Formally, an I/O automaton is defined as the following five-tuple: a signature (input, output, and internal actions), a set of states, a set of start states, a state-transition relation (a cross-product between states, actions, and states), and a partition of output and internal actions into *tasks*. Tasks are used for defining fairness conditions.

An action π is said to be *enabled* in a state s if the automaton has a transition of the form (s, π, s′); input actions are enabled in every state. An *execution* of an automaton is an alternating sequence of states and actions that begins with its start state and in which every action is enabled in the preceding state. An infinite execution is *fair* if, for each task, it contains either infinitely many actions from this task or infinitely many occurrences of states in which no action from this task is enabled; a finite execution is *fair* if no action is enabled in its final state. A *trace* is a subsequence of an execution solely consisting of the automaton's external actions. A *fair trace* is

a trace of a fair execution.

When reasoning about an automaton, we are interested in only its externally observable behavior as reflected in its traces. There are two types of trace properties: *safety* and *liveness*. Safety properties usually specify that some particular bad thing never happens. In this paper we specify safety properties using centralized, global, I/O automata that generate the legal sets of traces; for such automata we do not specify task partitions. Each external action in such a centralized automaton is tagged with a subscript which denotes the process at which this action occurs. An algorithm automaton *satisfies* a specification if all of its traces are also traces of the specification automaton. Refinement mappings are a commonly used technique for proving trace inclusion, in which one automaton (the algorithm) *simulates* the behavior of another automaton (the specification). Refinement mappings and other related proof techniques are reviewed in Appendix A. Liveness properties usually specify that some good thing eventually happens. An algorithm automaton satisfies a liveness property if the property holds in all of its *fair* traces.

The *composition operation* defines how automata interact via their input and output actions: It matches output and input actions with the same name in different component automata; when a component automaton performs a step involving an output action, so do all components that have this action as an input one. When reasoning about a certain system component, we compose it with abstract specification automata that specify the behavior of its environment.

I/O automata are conveniently presented using the *precondition-effect* style: In this style, typed state variables with initial values specify the set of states and the start states. A variable type is a set; if S is a set, the notation $S_\perp$ refers to the set $S \cup \{\perp\}$. Transitions are grouped by action name and are specified as a list of triples consisting of an action name possibly with parameters, a `pre` : block with preconditions on the states in which the action is enabled, and an `eff` : block which specifies how the pre-state is modified *atomically* to yield the poststate. The precondition-effect style is also known as a *guarded command* style: events have guards, or preconditions, and are triggered when the preconditions are enabled.

We have developed a novel formal notion of inheritance for automata [30]. A *child* automaton is specified as a modification of the parent automaton's code. When presenting a child we first specify a *signature extension* which consists of new actions, labeled new, and modified actions. A modified action is labeled with the name of the action which it modifies as follows: modifies `parent.action(parameters)`. We next specify the *state extension* consisting of new state variables added by the child. Finally, we describe the *transition restriction* which consists of new preconditions and effects added by the child to both new and modified actions. For modified actions, the preconditions and effects of the parent are appended to those added by the child. New effects added by the child are performed before the effects of the parent, all of them in a single atomic step. The child's effects are not allowed to modify state variables of the parent. This ensures that the set of traces of the child, when projected onto the parent's signature, is a subset of the parent's set of traces [30].

Inheritance allows us to reuse code and avoid redundancies. It also allows us to reuse proofs: Assume that an algorithm automaton `A` can simulate a specification automaton `S`, and let `A′` and `S′` be child automata of `A` and `S`, respectively. Then the proof extension theorem of [30] asserts that in order to prove that `A′` can simulate `S′`, it is sufficient to show that the restrictions added by `A′` are consistent with the restrictions `S′` places on `S` and that the new functionality of `A′` can simulate new

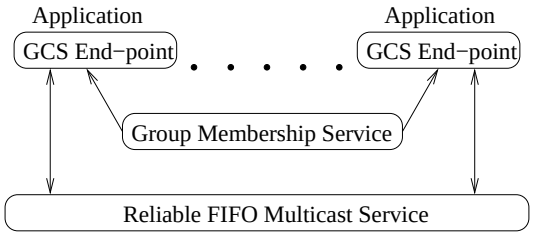


FIG. 4.1. *The client-server architecture: GCS end-points using an external membership service. Arrows represent interaction between GCS end-points and underlying services.*

functionality of $\mathtt{S}'$. Appendix A contains more details.

4. Client-server architecture and environment specification. Our service is designed to operate in an asynchronous message-passing environment. Processes and communication links may fail and may later recover, possibly causing network partitions and merges. For simplicity, we assume that processes recover with their running state intact; this is a plausible assumption as processes can keep their running state on stable storage. We do not explicitly model process crashes and recoveries because under this assumption a crashed process is indistinguishable from a slow one. In section 6.4, we argue that our algorithm also provides meaningful semantics when group communication processes lose their entire state upon a crash and recover with their state reset to an initial value.

Our GCS is implemented by a collection of GCS *end-points*, which are the GCS processes that run at the application clients' locations. GCS end-points handle clients' multicast requests and inform their clients of view changes.

The GCS architecture is depicted in Figure 4.1. All GCS end-points run the same algorithm. The algorithm relies on the underlying membership and multicast services to handle, respectively, formation of views and transmission of messages. The algorithm's task is to synchronize output of the two underlying services to implement the VS semantics.

Sections 4.1 and 4.2 below give precise specifications of the interface and semantics that the underlying membership and multicast services have to provide in order to be suitable for our algorithm. Services that satisfy these (or very similar) requirements have been previously used for GCSs, and efficient implementations of these services for WANs exist; see, for example, [31, 7].

4.1. The membership service specification. This section presents a formal specification of the membership services that are appropriate for our GCS design. For simplicity, here and in the rest of the paper, we assume that there is a single process group; multiple groups can be supported by treating each independently. We also omit part of the interface that handles processes' requests to join and leave groups.

Figure 4.2 contains an I/O automaton, called MEMB, that defines the interface and the safety properties of the membership service. The service interface is given by the automaton's signature.[3] Informally, it consists of the following two output

[3]When specifying a distributed system as a centralized automaton, we subscript each external action of the specification automaton with the location (or process) in the distributed system at which the action occurs.

AUTOMATON MEMB

Type:
Proc: Set of end-points.
StartId: Total-order; $\texttt{cid}_0$ is smallest.
ViewId: Partial-order; $\texttt{vid}_0$ is smallest.
View: ViewId × SetOf(Proc) × (Proc → StartId).
Def: $\texttt{v}_\texttt{p} = \langle \texttt{vid}_0, \{\texttt{p}\}, \{(\texttt{p} \rightarrow \texttt{cid}_0)\}\rangle$.

Signature:
Output: $\texttt{start}_\texttt{p}$(cid, set), Proc p, StartId cid, SetOf(Proc) set
$\texttt{view}_\texttt{p}$(v), Proc p, View v

State:
For all Proc p: View memb_view[p], initially $\texttt{v}_\texttt{p}$
For all Proc p: (StartId × SetOf(Proc)) start[p], initially $\langle \texttt{cid}_0, \{\}\rangle$

Transitions:

OUTPUT $\textbf{start}_\texttt{p}$(cid, set)
pre: cid > memb_view[p].startId(p)
cid ≥ start[p].id
p ∈ set
eff: start[p] ← ⟨cid, set⟩

OUTPUT $\textbf{view}_\texttt{p}$(v)
pre: p ∈ v.set ∧ v.id > memb_view[p].id
v.set ⊆ start[p].set
v.startId(p) = start[p].id
v.startId(p) > memb_view[p].startId(p)
eff: memb_view[p] ← v

FIG. 4.2. *Membership service interface and safety specification.*

actions:

$\texttt{start}_\texttt{p}$(cid, set) notifies process p that the membership service is attempting to form a view with the members of set; cid is a local start identifier;

$\texttt{view}_\texttt{p}$(v) notifies process p that the membership service has succeeded in forming view v. A view v is a triple consisting of an identifier v.id, a set of members v.set, and a function v.startId that maps members of v to start identifiers. Two views are the same if they consist of identical triples.

Automaton MEMB maintains two state variables, memb_view[p] and start[p], for each client p. These variables contain, respectively, the last view and the last start message issued to client p; the variables are updated in the effects of the transitions. The safety properties satisfied by the MEMB automaton include two basic properties, which are provided by virtually all group membership services (for example, [13, 20, 5, 23, 9, 31, 40, 3]), as well as some new properties concerning the start notifications.

The two basic properties are *self-inclusion* and *local monotonicity*. Self-inclusion requires every view issued to a client p to include p as a member; this property is enforced with a precondition p ∈ v.set on the $\texttt{view}_\texttt{p}$(v) action. Local monotonicity requires that view identifiers delivered to p be monotonically increasing; this property is enforced with a precondition v.id > memb_view[p] on the $\texttt{view}_\texttt{p}$(v) action. Local monotonicity has two important consequences: the same view is not delivered more than once to the same client, and clients that receive the same two views receive them in the same order [16].

In addition, the MEMB automaton specifies that the membership service must issue at least one start notification to client p before issuing a new view v to p. Also, the start identifier v.startId(p) contained in the new view v must be the same as the identifier of the latest preceding start issued to p. These two requirements are enforced by the last two preconditions on $\texttt{view}_\texttt{p}$(v). In particular, the former one is achieved by requiring that a bigger start identifier than the one associated with p in the last view has been issued to p.

The MEMB specification allows the membership service to react to connectivity changes happening during view formation. Whenever the service wants to add new

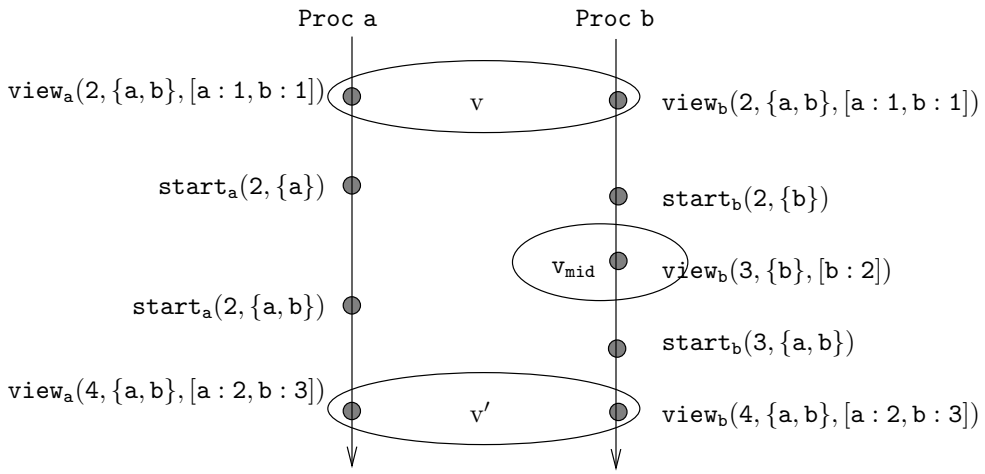


FIG. 4.3. *A sample execution of* MEMB.

members to the membership, it has to issue a new `start` notification to the clients: the second precondition on $\texttt{view}_{\texttt{p}}(\texttt{v})$ actions requires the membership `v.set` to be a subset of the tentative membership `set` included in the last `start` notification. In order to remove members from a forming view, the service does not need to issue a new `start` notification.

The first `start` notification issued to `p` after a view marks the beginning of a new view formation period. It includes a new local identifier `cid`, different from the ones that were previously sent to `p`: the first precondition on $\texttt{start}_{\texttt{p}}(\texttt{cid}, \texttt{set})$ requires `cid` to be strictly greater than `memb_view`[p].`startId`(p). Subsequent `start` notifications sent during an ongoing view formation may either reuse the last start identifier or issue a new one, as specified by the second precondition on `start` actions. We ensure uniqueness of local start identifiers by generating them in increasing order.

Notice that the MEMB automaton does not specify any relationship between views issued to different clients.

EXAMPLE 4.1. *Figure* 4.3 *presents a sample execution that shows the* MEMB *service delivering different sequences of views to two different clients,* a *and* b. *Arrows represent time passage at each client; gray dots represent events. First, both clients receive the same view* $\texttt{v} = \langle 2, \{\texttt{a}, \texttt{b}\}, [\texttt{a} : 1, \texttt{b} : 1]\rangle$*; we illustrate this with a circle around the view events at both clients. Then, client* b *receives a view* $\texttt{v}_{\texttt{mid}} = \langle 3, \{\texttt{b}\}, [\texttt{b} : 2]\rangle$ *by itself. Then, both clients receive another common view* $\texttt{v}' = \langle 4, \{\texttt{a}, \texttt{b}\}, [\texttt{a} : 2, \texttt{b} : 3]\rangle$*. Notice how the start identifiers included in the views correspond to the last start identifiers issued to the clients.*

We do *not* specify liveness properties for membership services. Instead, when we specify the liveness properties of our GCS in section 5.2, we *condition* them on the behavior of the membership service. For example, we state that if the same view is delivered to all the members and the members do not receive any subsequent membership events, then they eventually deliver this view to their application clients. Existing membership services do satisfy meaningful liveness properties. For example, [31] guarantees that, when the network stabilizes, all members receive the "correct" view and no other views thereafter. By combining our GCS liveness properties with such membership liveness properties, we can restate the liveness properties of our GCS

AUTOMATON CO_RFIFO

Signature:

Input:
  $\texttt{send}_{\texttt{p}}$(set,m), Proc p, SetOf(Proc) set, Msg m
  $\texttt{reliable}_{\texttt{p}}$(set), Proc p, SetOf(Proc) set
  $\texttt{live}_{\texttt{p}}$(set), Proc p, SetOf(Proc) set

Output: $\texttt{deliver}_{\texttt{p,q}}$(m), Proc p, Proc q, Msg m

Internal: lose(p,q), Proc p, Proc q
  skip_task(p,q), Proc p, Proc q

State:

For all Proc p, Proc q: SequenceOf(Msg) channel[p][q], initially empty
For all Proc p: SetOf(Proc) reliable_set[p], initially {p}
For all Proc p: SetOf(Proc) live_set[p], initially {p}

Transitions:

INPUT $\textbf{send}_{\texttt{p}}$(set, m)
eff: (∀ q ∈ set) append m to channel[p][q]

OUTPUT $\textbf{deliver}_{\texttt{p,q}}$(m)
pre: m = first(channel[p][q])
eff: dequeue m from channel[p][q]

INPUT $\textbf{reliable}_{\texttt{p}}$(set)
eff: reliable_set[p] ← set

INTERNAL **lose**(p, q)
pre: q ∉ reliable_set[p]
eff: dequeue last message from channel[p][q]

INPUT $\texttt{live}_{\texttt{p}}$(set)
eff: live_set[p] ← set

INTERNAL skip_task(p, q)
pre: q ∉ live_set[p]

Tasks:

For each Proc p, Proq q: $C_{\texttt{p,q}} = (\{deliver_{\texttt{p,q}}(m) \mid m \in Msg\} \cup \{skip\_task(p,q)\} \cup \{lose(p,q)\})$

FIG. 4.4. *Reliable* FIFO *multicast service specification. Liveness-related code is italicized.*

conditionally on the network behavior.

The MEMB specification allows for simple and efficient distributed implementations that also satisfy meaningful liveness properties. The membership service of [31] is an example of such an implementation; our design was implemented by Tarashchanskiy [43] using this membership service. In this service, a small number of servers support a large number of clients, communicating with them asynchronously via FIFO ordered channels (TCP sockets). In case a server fails, clients can migrate to another server. Other existing membership algorithms (for example, [20, 5]) could also be easily extended to provide the interface and semantics specified here.

4.2. The reliable FIFO multicast service specification. The group communication end-points communicate with each other using an underlying multicast service that provides reliable FIFO communication between every pair of connected processes. Many existing group communication systems (for example, [26, 9, 20, 3]) implement VS over similar communication substrates. In our implementation [43], we use the service of [7].

Figure 4.4 presents an I/O automaton, CO_RFIFO, that specifies a multicast service appropriate for our GCS design. Portions of the code that define liveness properties are italicized.

Automaton CO_RFIFO maintains a FIFO queue $\texttt{channel}[\texttt{p}][\texttt{q}]$ for every pair of end-points. An input action $\texttt{send}_{\texttt{p}}(\texttt{set}, \texttt{m})$ models a multicast of message m from end-point p to the end-points listed in the set by appending m to the $\texttt{channel}[\texttt{p}][\texttt{q}]$ queues for every end-point q in set. The $\texttt{deliver}_{\texttt{p,q}}(\texttt{m})$ action removes the first message from $\texttt{channel}[\texttt{p}][\texttt{q}]$ and delivers it to q.

In addition, the interface of CO_RFIFO includes input actions of the type $\texttt{reliable}_{\texttt{p}}(\texttt{set})$; end-point p may use such actions to command the multicast service to maintain a reliable (gap-free) FIFO connection to the end-points listed in set. Whenever this action occurs, set is stored in a special variable $\texttt{reliable\_set}[\texttt{p}]$. For every process q not in $\texttt{reliable\_set}[\texttt{p}]$, the multicast service may lose an arbitrary suffix of the messages sent from p to q, as modeled by an internal action $\texttt{lose}(\texttt{p}, \texttt{q})$.

In order for the multicast service to be considered live, messages sent to live and connected processes must eventually reach their destinations. The CO_RFIFO specification enforces this property in the italicized portion of its code.

Recall from section 3 that an infinite fair execution of an automaton must contain either infinitely many events from each task $\mathtt{C}$ or infinitely many occurrences of states in which no action in $\mathtt{C}$ is enabled. Automaton CO_RFIFO defines the set $\mathtt{C}_{\mathtt{p},\mathtt{q}} = (\{\mathtt{deliver}_{\mathtt{p},\mathtt{q}}(\mathtt{m}) \mid \mathtt{m} \in \mathtt{Msg}\} \cup \mathtt{skip\_task}(\mathtt{p},\mathtt{q}) \cup \mathtt{lose}(\mathtt{p},\mathtt{q}))$ to be a task for each pair of end-points $\mathtt{p}$ and $\mathtt{q}$. This definition implies that $\mathtt{deliver}_{\mathtt{p},\mathtt{q}}$ actions must occur in an infinite fair execution of CO_RFIFO, provided the following three conditions hold: there are messages sent from $\mathtt{p}$ to $\mathtt{q}$—hence, $\mathtt{deliver}_{\mathtt{p},\mathtt{q}}$ is enabled; the client at $\mathtt{p}$ is interested in maintaining reliable connection to $\mathtt{q}$—hence, $\mathtt{lose}(\mathtt{p},\mathtt{q})$ is disabled; and $\mathtt{q}$ is believed to have a live connection to $\mathtt{p}$—hence, a special action $\mathtt{skip\_task}(\mathtt{p},\mathtt{q})$ is disabled, as explained below.

Action $\mathtt{skip\_task}(\mathtt{p},\mathtt{q})$ is defined only to provide an alternative to $\mathtt{deliver}_{\mathtt{p},\mathtt{q}}$ actions so that $\mathtt{deliver}_{\mathtt{p},\mathtt{q}}$ actions are not required to happen when $\mathtt{q}$ is believed to be disconnected from $\mathtt{p}$. $\mathtt{skip\_task}(\mathtt{p},\mathtt{q})$ is an internal action that has no effect on the state of CO_RFIFO and is enabled when $\mathtt{q}$ is believed to be disconnected from $\mathtt{p}$. Such belief is modeled using special $\mathtt{live}_{\mathtt{p}}(\mathtt{set})$ input actions. The $\mathtt{set}$ argument is assumed to represent a set of processes that are alive and connected to $\mathtt{p}$; when such an input happens, $\mathtt{set}$ is stored in a state variable $\mathtt{live\_set}[\mathtt{p}]$. The precondition on the $\mathtt{skip\_task}(\mathtt{p},\mathtt{q})$ action is $\mathtt{q} \notin \mathtt{live\_set}[\mathtt{p}]$.

An important implication of how tasks are defined in CO_RFIFO is that, if $\mathtt{q}$ remains in both $\mathtt{live\_set}[\mathtt{p}]$ and $\mathtt{reliable\_set}[\mathtt{p}]$ from some point on in a fair execution of CO_RFIFO, then all the messages that $\mathtt{p}$ sends to $\mathtt{q}$ from that point on are eventually delivered to $\mathtt{q}$.

5. Specifications of the group communication service. The next two subsections contain specifications of the safety and liveness properties satisfied by our GCS. The specifications capture a core set of properties that is commonly provided by GCSs and that have been shown to be useful for facilitating implementations of many distributed applications and other, stronger, group communication properties (see [16]). For example, [32, Chapter 10] illustrates the utility of our GCS system by describing a simple application that can be effectively built using GCS. The application implements a variant of a data service that allows a dynamic group of clients to access and modify a replicated data object.

5.1. Safety properties. We present the safety specification of our GCS incrementally as four automata: In section 5.1.1 we specify a simple GCS that synchronizes delivery of views and application messages to require *within-view delivery* of messages. In section 5.1.2 we extend the specification of section 5.1.1 to also require *virtually synchronous delivery*, the key property of VS (see section 1.1). In section 5.1.3 we specify the *transitional set* property, which complements virtually synchronous delivery. Finally, in section 5.1.4, we specify the *self-delivery* property, which requires the GCS to deliver to each client the client's own messages.

The incremental development of the safety specification is matched later when we develop the algorithm and its correctness proof in section 6 and Appendix B.

5.1.1. Within-view reliable FIFO multicast. In this section we specify a GCS that captures the following properties:

1. Views delivered to the application satisfy the self-inclusion and local monotonicity properties of the MEMB service; see section 4.1.

AUTOMATON WV_RFIFO : SPEC

Signature:
Input: $\text{send}_p(m)$, Proc p, AppMsg m
Output: $\text{deliver}_p(q, m)$, Proc p, Proc q, AppMsg m
$\text{view}_p(v)$, Proc p, View v

State:
For all Proc p, View v: SequenceOf(AppMsg) msgs[p][v], initially empty
For all Proc p, Proc q: Int last_dlvrd[p][q], initially 0
For all Proc p: View current_view[p], initially v_p

Transitions:
INPUT $\textbf{send}_p(m)$
eff: append m to msgs[p][current_view[p]]

OUTPUT $\textbf{deliver}_p(q, m)$
pre: m = msgs[q][current_view[p]][last_dlvrd[q][p]+1]
eff: last_dlvrd[q][p] ← last_dlvrd[q][p]+1

OUTPUT $\textbf{view}_p(v)$
pre: p ∈ v.set ∧ v.id > current_view[p].id
eff: (∀ q) last_dlvrd[q][p] ← 0
current_view[p] ← v

FIG. 5.1. WV_RFIFO *service specification.*

2. Messages are delivered in the same view in which they were sent. This property is useful for many applications (see [23, 16, 42]) and appears in several systems and specifications (for example, [13, 44, 5, 38, 22, 28, 18]). A weaker property that requires each message to be delivered in the same view at every process that delivers it, but not necessarily the view in which it was sent, is typically implemented on top of an implementation of within-view delivery (see [16]).

3. Messages are delivered in gap-free FIFO order (within views). This is a basic property upon which one can build services with stronger ordering guarantees, such as causal order or total order. The totally ordered multicast algorithm of [14] is implemented atop a service with a similar specification.

Figure 5.1 presents automaton WV_RFIFO : SPEC that models this specification. The automaton uses centralized queues `msgs`[p][v] of application messages for each sender `p` and view `v`. It also maintains a variable `current_view`[p] that contains the last view delivered to each process `p` and a variable `last_dlvrd`[q][p], for every pair of processes `q` and `p`, containing the index in the `msgs`[q][`current_view`[p]] queue of the last message from `q` delivered to `p` in `p`'s current view.

Action $\text{view}_p(v)$ models the delivery of view `v` to process `p`; the precondition on this action enforces self-inclusion and local monotonicity. Action $\text{send}_p(m)$ models the multicast of message `m` from process `p` to the members of `p`'s current view by appending `m` to `msgs`[p][`current_view`[p]]. Action $\text{deliver}_p(q, m)$ models the delivery to process `p` of message `m` sent by process `q`. The gap-free FIFO ordered delivery of messages within-views is enforced by its precondition, which allows delivery of only the message indexed by `last_dlvrd`[q][p] + 1 in the `msgs`[q][`current_view`[p]] queue.

5.1.2. Virtually synchronous delivery. In this section we use the inheritance-based methodology to modify the WV_RFIFO : SPEC automaton to also enforce the *virtually synchronous delivery* property. The modified automaton, VSRFIFO : SPEC, is defined by the code contained in both Figures 5.1 and 5.2.

Figure 5.2 contains the code that enforces the *virtually synchronous delivery* property. Recall from section 1.1 that this property requires processes moving together from view `v` to view `v′` to deliver same set of messages while in view `v`. Since the parent specification, WV_RFIFO : SPEC, imposes gap-free FIFO delivery of messages, a message set can be represented by a set of indices, each pointing to the last message from each member of `v`; such representation of a set is called a *cut*.

AUTOMATON VS_RFIFO : SPEC MODIFIES WV_RFIFO : SPEC

Signature Extension:
Output: $\texttt{view}_p(v)$ modifies $\texttt{wv\_rfifo.view}_p(v)$
Internal: set_cut(v, v′, c), View v, View v′, $(\text{Proc} \to \text{Int})_\perp$ c new

State Extension:
For all View v, v′: $(\text{Proc} \to \text{Int})_\perp$ cut[v][v′], initially $\perp$

Transition Restriction:

OUTPUT $\textbf{view}_p(v)$
pre: cut[current_view[p]][v] $\neq \perp$
(∀ q) last_dlvrd[q][p]=cut[current_view[p]][v](q)

INTERNAL **set_cut**(v, v′, c)
pre: cut[v][v′] $= \perp$
eff: cut[v][v′] $\leftarrow$ c

FIG. 5.2. VS_RFIFO *service specification.*

The WV_RFIFO : SPEC automaton fixes a cut for processes that wish to move from some view v to some view v′: A new internal action $\texttt{set\_cut}(v, v', c)$ sets a new variable $\texttt{cut}[v][v']$ to a cut mapping c. For a given pair of views, v and v′, the cut is chosen only once, *nondeterministically.* Delivery of a view v to process p is allowed only if a cut for moving from p's current view into v has been set and if p has delivered all the messages identified in this cut. These conditions are enforced by the two new preconditions of the $\texttt{view}_p(v)$ action (see Figure 5.2). Since VSRFIFO : SPEC is a modification of WV_RFIFO : SPEC, the new preconditions work in conjunction with the preconditions in $\texttt{view}_p(v)$ of WV_RFIFO : SPEC.

The VSRFIFO : SPEC automaton, being a safety specification, does not require liveness properties to hold, for instance, that processes actually deliver messages specified by the cuts and hence are able to satisfy conditions for delivering new views. Such liveness specifications are stated in section 5.2.

5.1.3. Transitional set. While virtually synchronous delivery is a useful property, a process that moves from view v to view v′ cannot tell locally which of the processes in $v.\texttt{set} \cap v'.\texttt{set}$ move to view v′ directly from view v and which move to v′ from some other view. In order for the application to be able to exploit the virtually synchronous delivery property, application processes need to be informed which other processes move together with them from their current view into their new view. The set of processes that transition together from one view into the next is called a *transitional set* [16].

DEFINITION 5.1. *A transitional set from view* v *to view* v′ *is a subset of* $v.\texttt{set} \cap v'.\texttt{set}$ *that includes* (a) *all processes that receive view* v′ *while in view* v *and* (b) *no process that receive view* v′ *while in a view other than* v.

Note that the transitional set is not uniquely defined by Definition 5.1. If a process p in $v.\texttt{set} \cap v'.\texttt{set}$ does not receive view v′, Definition 5.1 does not specify whether or not p is included in the transitional sets of other processes that do receive view v′.

The notion of a transitional set was first introduced as part of a special transitional view in the EVS [38] model. In our formulation (as in [16]), transitional sets are delivered to the application along with views as an additional parameter T.

EXAMPLE 5.1. *Assume that Alice and Bob are using a virtually synchronous GCS that eventually reports the views produced by the* MEMB *service to Alice and Bob. Consider the scenario described in Example* 4.1*: both Alice and Bob receive views* v *and* v′ *with the membership* {*Alice, Bob*}*. Just from these views, Alice does not know whether Bob receives view* v′ *while in view* v *or while in some other view* v_{mid} *with the membership* {*Bob*}*. If the former holds, then Alice does not need to synchronize with Bob because virtually synchronous delivery guarantees that they have received the same messages while in view* v*; otherwise, she does. The transitional set*

AUTOMATON TRANS_SET : SPEC

Signature:
Output: $\texttt{view}_p(v,T)$, Proc p, View v, SetOf(Proc) T
Internal: $\texttt{set\_prev\_view}_p(v)$, Proc p, View v

State:
For all Proc p: View current_view[p], initially v_p
For all Proc p, View v: $\text{View}_\perp$ prev_view[p][v], initially $\perp$

Transitions:

OUTPUT **view_p(v, T)**
pre: prev_view[p][v] = current_view[p]
(∀ q ∈ v.set ∩ current_view[p].set)
prev_view[q][v] ≠ ⊥
T = {q ∈ v.set ∩ current_view[p].set |
prev_view[q][v] = current_view[p]}
eff: current_view[p] ← v

INTERNAL **set_prev_view_p(v)**
pre: p ∈ v.set
prev_view[p][v] = ⊥
eff: prev_view[p][v] ← current_view[p]

FIG. 5.3. *Transitional set specification.*

AUTOMATON WV_RFIFO+SELF : SPEC MODIFIES WV_RFIFO : SPEC

Signature Extension:
Output: $\text{view}_p(v)$ modifies $\text{wv\_rfifo.view}_p(v)$

Transition Restriction:
OUTPUT **view_p(v)**
pre: last_dlvrd[p][p] = LastIndexOf(msgs[p][current_view[p]])

FIG. 5.4. WV_RFIFO+SELF *service specification.*

given to Alice together with view v' *provides this information.*

Figure 5.3 presents an automaton TS : SPEC that specifies delivery of transitional sets (Definition 5.1). There are two types of actions: output actions $\texttt{view}_p(v, T)$ deliver view v and transitional set T to process p; and internal actions $\texttt{set\_prev\_view}_p(v)$ declare that q intends to deliver view v while in its current view. The intentions are recorded in the variable prev_view[p][v], and the current views are recorded in the variable current_view[p].

Before process p can deliver a view v, each member q in the intersection of these views must execute $\texttt{set\_prev\_view}_q(v)$, as enforced by the second precondition. The transitional set T delivered by p with v is then computed to consist of those processes q in the intersection current_view[p].set ∩ v.set for which prev_view[q][v] is the same as current_view[p]; this is specified by the third precondition on $\texttt{view}_p(v, T)$.

5.1.4. Self-delivery. We now specify the *self-delivery* property, which requires that each client receives all the messages it sent in a given view before receiving a new view. We specify this property as a simple modification of the WV_RFIFO : SPEC automaton presented in section 5.1.1; the modified automaton is defined by the code contained in both Figures 5.1 and 5.4.

In order to enforce self-delivery, a new precondition on the $\texttt{view}_p(v)$ action requires the last_dlvrd[p][p] index to point to the last message sent by client p in its current view. Since the parent automaton, WV_RFIFO : SPEC, guarantees within-view gap-free FIFO delivery, this precondition implies that all of p's messages have in fact been delivered back to p.

In order for a GCS to be live and satisfy within-view delivery, self-delivery, and virtually synchronous delivery, the GCS must *block* its application from sending new messages during view formation periods; this is proved in [23]. Therefore, we introduce a block/block_ok synchronization when we extend our algorithm to support the self-

delivery property in section 6.3.

Our formulation of self-delivery as a safety property, when combined with the liveness property of section 5.2, implies the formulations in [16] and [38] of self-delivery as a liveness property. These formulations require a GCS to *eventually* deliver to each process its own messages.

5.2. Liveness property. In a fault-prone asynchronous model, it is not feasible to require that a GCS be live in every execution. The only way to specify useful liveness properties without strengthening the communication model is to make these properties *conditional* on the underlying network behavior (as specified, for example, in [22, 17, 16]). Since our GCS uses an external membership service, we condition the GCS liveness on the behavior of the membership service.

We define the liveness property for a restricted set of executions in which a component stabilizes from some point on forever thereafter.

PROPERTY 5.1 (view stability). *Let* GCS *be a group communication service whose interface with its clients consists of* `send`, `deliver`, *and* `view` *events as defined in the automaton signature in Figure* 5.1. *Furthermore, assume that the* GCS *uses a membership service* MEMB *described in section* 4.

A view $\mathtt{v}$ *eventually becomes* stable *in a given timed execution* $\alpha = \mathtt{s}_0, \pi_1, \mathtt{s}_1, \pi_2, \ldots$ *of the* GCS *service, in the sense that a* MEMB.$\mathtt{view}_\mathtt{p}(\mathtt{v})$ *event occurs in* α *for every* $\mathtt{p} \in \mathtt{v.set}$ *and is followed by neither* MEMB.$\mathtt{view}_\mathtt{p}$ *nor* MEMB.$\mathtt{start}_\mathtt{p}$ *events.*

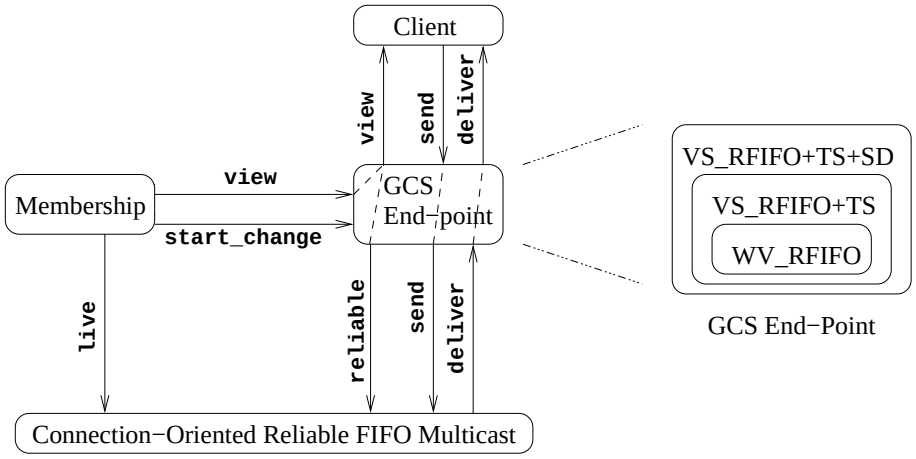
Given an execution that satisfies Property 5.1, the liveness property requires each end-point in the stable view to eventually deliver this last view and all the messages sent in this view to its client. Formally, we have the following property.

PROPERTY 5.2 (liveness). *Let* GCS *be a group communication service whose interface with its clients consists of* `send`, `deliver`, *and* `view` *events as defined in the automaton signature in Figure* 5.1. *Furthermore, assume that the* GCS *uses a membership service* MEMB *described in section* 4.

Let α *be a fair execution of* GCS *in which view* $\mathtt{v}$ *eventually becomes stable (Property* 5.1*). Then, at each* $\mathtt{p} \in \mathtt{v.set}$, GCS.$\mathtt{view}_\mathtt{p}(\mathtt{v})$ *eventually occurs. Moreover, for every* GCS.$\mathtt{send}_\mathtt{p}(\mathtt{m})$ *that occurs after* GCS.$\mathtt{view}_\mathtt{p}(\mathtt{v})$, *and for every* $\mathtt{q} \in \mathtt{v.set}$, GCS.$\mathtt{deliver}_\mathtt{q}(\mathtt{p}, \mathtt{m})$ *also occurs.*

It is important to note that although our liveness property requires the GCS to be live only in *certain* executions, any implementation that satisfies this property has to attempt to be live in *every* execution because it cannot test the external condition of the membership becoming stable. Also note that, even though membership stability is formally required to last forever, in practice it only has to hold "long enough" for the GCS to reconfigure, as explained in [21, 25]. However, we cannot explicitly introduce the bound on this time period in a fully asynchronous model, since it depends on external conditions such as message latency, process scheduling, and processing time.

6. The virtually synchronous group multicast algorithm. In this section we present an algorithm for a group communication service, GCS, that satisfies the specifications in section 5. The GCS is implemented by a collection of GCS end-points, each running the same algorithm. Figure 6.1(a) shows the interaction of a GCS end-point with its environment: a membership service MEMB and a reliable FIFO multicast service CO_RFIFO; these services are assumed to satisfy the specifications of section 4. The end-point interacts with its application client by accepting the client's send requests and by delivering application messages and views to the client. The end-point uses the CO_RFIFO service to send messages to other GCS end-points and to receive messages sent by other GCS end-points. When necessary, the end-point uses



(a) Interaction with Membership and Multicast Services **(b)** Inheritance Hierarchy

FIG. 6.1. *A GCS end-point and its environment.*

the `reliable` action to inform CO_RFIFO of the set of end-points to which CO_RFIFO must maintain reliable (gap-free) FIFO connections. The GCS end-point also receives `start` and `view` notifications from the membership service.

The algorithm running at each GCS end-point is constructed incrementally using the inheritance-based methodology of [30]. We proceed in three steps, at each step adding support for a new property (see Figure 6.1(b)):

1. In section 6.1, we present an algorithm WV_RFIFO_p for an end-point of the within-view reliable FIFO multicast service specified in section 5.1.1 and argue that this service satisfies safety specification WV_RFIFO : SPEC and liveness Property 5.2.

2. In section 6.2, we add support for the virtually synchronous delivery and transitional set properties specified in sections 5.1.2 and 5.1.3. We present a child VS_RFIFO+TS_p of WV_RFIFO_p and argue that the service built from VS_RFIFO+TS_p end-points satisfies safety specifications VSRFIFO : SPEC and TS : SPEC and liveness Property 5.2.

3. In section 6.3, we add support for the self-delivery property specified in section 5.1.4. The resulting automaton VS_RFIFO+TS+SD_p models a complete GCS end-point. Due to the use of inheritance, the service built from these end-points automatically satisfies safety specifications WV_RFIFO : SPEC, VSRFIFO : SPEC, and TS : SPEC. We argue that it also satisfies safety specification SELF : SPEC and liveness Property 5.2.

In the presented automata, each locally controlled action is defined to be a task by itself, which means that, if it becomes and stays enabled, it eventually gets executed.

When composing automata into a service, actions of the $\text{MEMB.start}_p(\texttt{id}, \texttt{set})$ type are linked with $\text{CO\_RFIFO.live}_p(\texttt{set})$, and actions of the $\text{MEMB.view}_p(\texttt{v})$ type are linked with $\text{CO\_RFIFO.live}_p(\texttt{v.set})$; the "link" operation can be formally expressed using the signature extension construct. When MEMB and CO_RFIFO actions are linked this way, the `live_set`[`p`] variable of CO_RFIFO matches the MEMB's perception of which end-points are alive and connected to `p`. (We assume that every permanently disconnected end-point is eventually excluded by either a `start` or a `view` notifica-

tion.) In the composed system, all output actions except the application interface are reclassified as internal.

For simplicity of the code, the presented automata do not include certain practical optimizations such as, for example, garbage collection; we point out some of the important ones in section 6.4.

6.1. Within-view reliable FIFO multicast algorithm. In this section we present the WV_RFIFO$_p$ algorithm running at an end-point p of a basic GCS, WV_RFIFO. The end-point algorithm is quite simple: It relies on the MEMB service to form and deliver views involving end-point p; the end-point forwards these views to its client. The algorithm also relies on the CO_RFIFO service to provide reliable gap-free FIFO multicast communication. When the end-point receives a message-send request from its client, it uses CO_RFIFO to send the message to other end-points in the client's current view. The end-point delivers to its client the messages received from other end-points via CO_RFIFO, provided the client's current view matches the views in which the messages were sent. The algorithm keeps track of the views in which messages are sent using the following technique: each time the end-point delivers a view v to its client, it sends a special view_msg message to the end-points in v.set informing them that the end-point's future messages will be sent in view v. Reliable delivery of messages is ensured by having CO_RFIFO maintain a reliable connection to every member of the end-point's view.

Figure 6.2 models the WV_RFIFO$_p$ algorithm as an automaton. The signature defines the interface through which end-point p interacts with its client and with the MEMB and CO_RFIFO services.

When a view v is received from MEMB via action MEMB.view$_p$(v), end-point p saves it in a variable memb_view and then delivers v to its client by executing action view$_p$(v). Variable current_view contains the last view delivered to the client. The precondition, v = memb_view $\neq$ current_view, on the view$_p$(v) action ensures that v is indeed the last view received from MEMB and that it has not already been delivered to the client. After end-point p delivers view v to its client, it sends a view_msg containing v to the rest of the members of current_view.set by using action CO_RFIFO.send$_p$(set, ⟨'**view_msg**', v⟩) with set = current_view.set − {p} and v = current_view. Variable view_msg[p] contains the last view sent as a view_msg. The first precondition on CO_RFIFO.send$_p$(set, ⟨'**view_msg**', v⟩), view_msg[p] $\neq$ current_view, ensures that each view_msg is sent only once, and the second precondition, current_view.set $\subseteq$ reliable_set, ensures that, prior to sending the view_msg, end-point p has requested CO_RFIFO to maintain reliable connection to every member of the client's view by executing action CO_RFIFO.reliable$_p$(set), which sets variable reliable_set to the value of set. When end-point p receives a view_msg from some end-point q via the CO_RFIFO.deliver$_{q,p}$(⟨'**view_msg**', v⟩) action, it stores v in a variable view_msg[q].

End-point p maintains a queue msgs[q][v] per each end-point q and view v; these queues are used for storing application messages received from other end-points via CO_RFIFO.deliver$_{q,p}$ and from the end-point's own client via send$_p$. When action send$_p$(m) occurs, message m is appended to msgs[p][current_view]. The end-point maintains the following indices that enforce message handling in the order of their appearances in the msgs queues:

- last_sent points to the last application message m on msgs[p][current_view] that was sent using CO_RFIFO.send$_p$(set, ⟨'**app_msg**', m⟩);
- last_rcvd[q], for each end-point q, points to the last message m on queue

AUTOMATON WV_RFIFO$_p$

Type:
```
ViewMsg = View
FwdMsg  = Proc × View × AppMsg × Int
```

Signature:
```
Input:  send_p(m), AppMsg m
        co_rfifo.deliver_q,p(m), Proc q,
              (AppMsg + ViewMsg + FwdMsg) m
        memb.view_p(v), View v

Output: deliver_p(q, m), Proc q, AppMsg m
        co_rfifo.send_p(set, m), SetOf(Proc) set,
              (AppMsg + ViewMsg + FwdMsg) m
        co_rfifo.reliable_p(set), SetOf(Proc) set
        view_p(v), View v
```

State:
```
// Variables for handling application messages
For all Proc q, View v: SequenceOf(AppMsg⊥)
    msgs[q][v], initially empty
Int last_sent, initially 0
For all Proc q: Int last_rcvd[q],  initially 0
For all Proc q: Int last_dlvrd[q], initially 0

// Variables for handling views and view messages
View current_view, initially v_p
View memb_view,  initially v_p
For all Proc q: View view_msg[q], initially v_q

SetOf(Proc) reliable_set, initially v_p.set
```

Transitions:
```
INPUT  memb.view_p(v)
eff: memb_view ← v

OUTPUT  view_p(v)
pre: v =  memb_view ≠ current_view
eff: current_view ← v
     last_sent  ← 0
     (∀ q) last_dlvrd[q] ← 0

OUTPUT  co_rfifo.reliable_p(set)
pre: current_view.set ⊆ set
     reliable_set ≠ set
eff: reliable_set ← set

OUTPUT  co_rfifo.send_p(set, ⟨'view_msg', v⟩)
pre: view_msg[p] ≠ current_view
     current_view.set ⊆ reliable_set
     set =  current_view.set - {p}
     v = current_view
eff: view_msg[p] ← current_view

INPUT  co_rfifo.deliver_q, p(⟨'view_msg', v⟩)
eff: view_msg[q]  ← v
     last_rcvd[q] ← 0

INPUT  send_p(m)
eff: append m to msgs[p][current_view]

OUTPUT  deliver_p(q, m)
pre: m = msgs[q][current_view][last_dlvrd[q]+1]
eff: last_dlvrd[q] ← last_dlvrd[q] + 1

OUTPUT  co_rfifo.send_p(set, ⟨'app_msg', m⟩)
pre: view_msg[p] = current_view
     set =  current_view.set - {p}
     m = msgs[p][current_view][last_sent + 1]
eff: last_sent ← last_sent + 1

INPUT  co_rfifo.deliver_q,p(⟨'app_msg', m⟩)
eff: msgs[q][view_msg[q]][last_rcvd[q]+1] ←m
     last_rcvd[q] ← last_rcvd[q] + 1

OUTPUT  co_rfifo.send_p(set,⟨'fwd_msg',r,v,m,i⟩)
pre: (p ∉ set)   ∧   (m = msgs[r][v][i])

INPUT  co_rfifo.deliver_q,p(⟨'fwd_msg',r,v,m,i⟩)
eff:  msgs[r][v][i] ← m
```

FIG. 6.2. *Within-view reliable* FIFO *multicast end-point automaton.*

msgs[q][view_msg[q]] that was delivered to p by CO_RFIFO.deliver$_{q,p}$(⟨'**app_msg**', m⟩);

- last_dlvrd[q], for each end-point q, points to the last message m on queue msgs[q][current_view] that was delivered to p's client using CO_RFIFO.deliver$_p$(q, m).

The first precondition of CO_RFIFO.send$_p$(set, ⟨'**app_msg**', m⟩) ensures that a view_msg containing current_view has been already sent to everybody in set = current_view − {p}. The preconditions on sending view_msgs ensure that CO_RFIFO maintains a reliable connection to everyone in set at the time CO_RFIFO.send$_p$(set, ⟨'**app_msg**', m⟩) occurs.

Automaton WV_RFIFO$_p$ also implements auxiliary functionality that allows end-point p to forward an application message received from some end-point to some other end-points. Specifically, using CO_RFIFO.send$_p$(set, ⟨'**fwd_msg**', r, v, m, i⟩), end-point p can forward to some set of end-points the ith message, m, sent by the client at r in view v. In turn, when end-point p receives CO_RFIFO.deliver$_{q,p}$(⟨'**fwd_msg**', r, v, m, i⟩), it stores the forwarded message m in the ith location of the msgs[r][v] queue. The code of WV_RFIFO$_p$ does not specify a particular strategy for forwarding messages; the

strategy can be chosen nondeterministically. Such a strategy can be specified by more refined versions of the algorithm and/or by modifications of WV_RFIFO$_p$, as we do in the VS_RFIFO+TS$_p$ modification of the WV_RFIFO$_p$ automaton in section 6.2 below.

Leaving a certain level of nondeterminism at the parent automaton, with the intention of resolving it later at the child automaton, is a technique similar to the use of *abstract methods* or *pure virtual methods* in object-oriented methodology. We use the same technique in the CO_RFIFO.reliable$_p$(set) action when we require set to be a nondeterministic superset of current_view.set. The VS_RFIFO+TS$_p$ modification of WV_RFIFO$_p$ places additional preconditions on this action, thereby specifying precise values for the set argument.

The WV_RFIFO automaton resulting from the composition of all the end-point automata and the MEMB and CO_RFIFO automata models the WV_RFIFO service. The automaton satisfies the safety properties specified by WV_RFIFO : SPEC: it preserves the local monotonicity and self-inclusion properties of view deliveries guaranteed by the MEMB service; it also extends the gap-free FIFO ordered message delivery of CO_RFIFO with the within-view delivery property. The within-view delivery is achieved by delivering messages to the clients only if the views in which the messages were sent match the clients' current views.

Appendix B.1 contains a simulation from WV_RFIFO to WV_RFIFO : SPEC: Actions of automaton WV_RFIFO : SPEC involving view$_p$(v), send$_p$(m), and deliver$_p$(q, m) are simulated when WV_RFIFO takes the corresponding view$_p$(v), send$_p$(m), and deliver$_p$(q, m) actions. The steps of WV_RFIFO involving other actions correspond to empty steps of WV_RFIFO : SPEC. We define the following function R that maps every reachable state s of WV_RFIFO to a reachable state of WV_RFIFO : SPEC, where s[p].var denotes an instance of a variable var of end-point p in a state s:

```
R(s∈ReachableStates(WV_RFIFO)) = t∈ReachableStates(WV_RFIFO : SPEC), where
  for each Proc p, View v:       t.msgs[p][v]   =   s[p].msgs[p][v],
  for each Proc p, Proc q: t.last_dlvrd[p][q]   =   s[q].last_dlvrd[p],
  for each Proc p:          t.current_view[p]   =   s[p].current_view.
```

Lemma B.1 states that R is a refinement mapping from automaton WV_RFIFO to automaton WV_RFIFO : SPEC; the proof relies on a number of invariant assertions, stated and proved in Appendix B.1 as well.

The WV_RFIFO automaton also satisfies liveness (Property 5.2). Consider a fair execution in which each end-point p in v.set receives the same view v from the membership and no view events afterwards. Starting from the time the MEMB.view$_p$(v) action occurs, the view$_p$(v) action stays enabled; therefore it eventually happens due to the fairness of the execution. After view v is delivered to the clients, all messages sent in view v are also eventually delivered to the clients. This is due to the liveness property of CO_RFIFO, which guarantees that messages sent between live and connected end-points (as perceived by the membership service) are eventually delivered to their destinations. We prove these claims formally for the complete GCS algorithm in Appendix C.

6.2. Adding support for virtually synchronous delivery and transitional sets. The WV_RFIFO service of the previous section guarantees that each member p of a view v receives *some* prefix of the FIFO ordered stream of messages sent by every member q in v. In this section, we modify the WV_RFIFO$_p$ algorithm to yield an end-point VS_RFIFO+TS$_p$ of a service, VS_RFIFO+TS, that, in addition to the semantics provided by WV_RFIFO, guarantees that those members that transition from v into

AUTOMATON VS_RFIFO+TS$_p$ MODIFIES WV_RFIFO$_p$

Type: SyncMsg = StartId × View × (Proc → Int)

Signature Extension:

Input: memb.start$_p$(id, set), StartId id, SetOf(Proc) set new
co_rfifo.deliver$_{q,p}$(m), Proc q, SyncMsg m new

Output: deliver$_p$(q, m) modifies wv_rfifo.deliver$_p$(q, m)
view$_p$(v, T), SetOf(Proc) T modifies wv_rfifo.view$_p$(v)
co_rfifo.reliable$_p$(set), SetOf(Proc) set modifies wv_rfifo.co_rfifo.reliable$_p$(set)
co_rfifo.send$_p$(set, m), SetOf(Proc) set, SyncMsg m new
co_rfifo.send$_p$(set, m) modifies wv_rfifo.co_rfifo.send$_p$(set, m), FwdMsg m

Internal: set_cut$_p$() new

FIG. 6.3. *Virtually synchronous reliable* FIFO *multicast: signature extension.*

the same view v' receive not just *some* but *the same* prefix of the message stream sent by each member q in v. This is the virtually synchronous delivery property, the key property of VS semantics (see section 5.1.2). Overall, the VS_RFIFO+TS service satisfies the VSRFIFO : SPEC and TS : SPEC safety specifications, as well as liveness Property 5.2; we prove these claims, respectively, in Appendixes B.2, B.3, and C.

In a nutshell, here is how VS_RFIFO+TS$_p$ computes transitional sets and enforces virtually synchronous delivery: When end-point p is notified via start$_p$(cid, set) of the MEMB's attempt to form a new view, p sends via CO_RFIFO a synchronization message tagged with cid to every end-point in set. The synchronization message includes p's current view v and a mapping cut, such that cut(q) is the index of the last message from each q in v.set that p commits to deliver in view v.

End-point p may receive subsequent start$_p$(cid, set) notifications from MEMB. When such a notification includes a new cid, p sends a new synchronization message, with a freshly made cut, to the proposed set; otherwise, when the cid is the same as the last one, p simply forwards the last synchronization message to the joining end-points, that is, to the end-points of the current set that were not listed in the previously proposed membership.

Once p receives via view$_p$(v') a new view v' from MEMB and a synchronization message tagged with v'.startId(q) from each end-point q in v.set ∩ v'.set, p computes a transitional set from v to v' and decides on which messages it needs to deliver to its client in view v before delivering view v'. A transitional set T from v to v' is computed to include every client q in v.set ∩ v'.set whose synchronization message tagged with v'.startId(q) contains p's current view v. For each client r in v.set, end-point p decides to deliver all the messages of r that appear in the cut of the synchronization message of any member q of T. Section 6.2.1 describes two message-forwarding strategies that ensure p's ability to actually deliver all the messages it decides to deliver. After p delivers all these messages to its client, it then delivers to its client the new view v' along with the transitional set T.

Virtually synchronous delivery follows from the fact that all end-points transitioning from view v to v' consider the same synchronization messages, compute the same set T, and hence use the same data to decide which messages to deliver in view v before delivering view v'. Set T satisfies Definition 5.1 of a transitional set from v to v' because (a) every end-point that computes T is itself included in T, and (b) no end-point q in T is allowed to deliver v' while in some view other than v because v'.startId(q) is linked through q's synchronization message to v.

AUTOMATON VS_RFIFO+TS$_p$ MODIFIES WV_RFIFO$_p$

State Extension:

```
(StartId × SetOf(Proc))⊥ start, initially ⊥
For all Proc q, StartId id: (View v, (Proc → Int) cut)⊥ sync_msg[q][id], initially ⊥
SetOf(Proc) sync_set, initially empty
SetOf((Proc × Proc × View × Int)) forwarded_set, initially empty
```

Transition Restriction:

```
INPUT memb.start_p(cid, set)
eff: if start ≠ ⊥ ∧ start.id = cid
       then sync_set ← sync_set ∩ set
       else sync_set ← ∅
     start ← ⟨cid, set⟩

OUTPUT co_rfifo.reliable_p(set)
pre: start = ⊥ ⇒ set = current_view.set
     start ≠ ⊥ ⇒ set = current_view.set ∪ start.set

INTERNAL set_cut_p()
pre: start ≠ ⊥ ∧ sync_msg[p][start.id] = ⊥
eff: Let cut = {⟨q, LongestPrefixOf(msgs[q][current_view])⟩ | q ∈ current_view.set}
     sync_msg[p][start.id] ← ⟨current_view, cut⟩
     sync_set ← {p}

OUTPUT co_rfifo.send_p(set, ⟨'sync_msg', cid, v, cut⟩)
pre: start ≠ ⊥ ∧ sync_msg[p][start.id] ≠ ⊥
     set = (start.set - sync_set) ≠ ∅
     set ⊆ reliable_set
     cid = start.id ∧ ⟨v, cut⟩ = sync_msg[p][cid]
eff: sync_set ← start.set

INPUT co_rfifo.deliver_q,p(⟨'sync_msg', cid, v, cut⟩)
eff: sync_msg[q][cid] ← ⟨v, cut⟩

OUTPUT deliver_p(q, m)
pre: if (start ≠ ⊥ ∧ sync_msg[p][start.id] ≠ ⊥) then
       if start.id ≠ memb_view.startId(p) then
          last_dlvrd[q]+1 ≤ sync_msg[p][start.id].cut(q)
       else let S = {r ∈ memb_view.set ∩ current_view.set |
                     sync_msg[r][memb_view.startId(r)].view = current_view}
            last_dlvrd[q]+1 ≤ max_r∈S sync_msg[r][memb_view.startId(r)].cut(q)

OUTPUT view_p(v, T)
pre: v.startId(p) = start.id                  // to prevent delivery of obsolete views
     v.set - sync_set = ∅                     // all sync msgs are sent
     last_sent ≥ sync_msg[p][v.startId(p)].cut(p)   // sent out your own msgs
     (∀ q ∈ v.set ∩ current_view.set) sync_msg[q][v.startId(q)] ≠ ⊥
     T = {q ∈ v.set ∩ current_view.set | sync_msg[q][v.startId(q)].view = current_view}
     (∀ q ∈ current_view.set) last_dlvrd[q] = max_r∈T sync_msg[r][v.startId(r)].cut(q)
eff: start ← ⊥
     sync_set ← ∅

OUTPUT co_rfifo.send_p(set,⟨'fwd_msg',r,v,m,i⟩)
pre: (∀ q ∈ set) (⟨q, r, v, i⟩ ∉ forwarded_set)  ∧  ForwardStrategyPredicate(set, r, v, i)
eff: (∀ q ∈ set) add ⟨q, r, v, i⟩ to forwarded_set
```

FIG. 6.4. *Virtually synchronous reliable* FIFO *multicast: state extension and transition restriction.*

Figures 6.2, 6.3, and 6.4, together, contain the code of the VS_RFIFO+TS$_p$ automaton that models end-point p of the VS_RFIFO+TS service. Figures 6.3 and 6.4 specify how the WV_RFIFO$_p$ automaton of Figure 6.2 is modified to support virtually synchronous delivery and transitional sets. Figure 6.3 contains signature extension that defines the signatures of new and modified actions; Figure 6.4 contains the state extension and transition restriction defining, respectively, new state variables and new precondition/effect code. We now describe automaton VS_RFIFO+TS$_p$ in detail.

Upon receiving MEMB.start$_p$(cid, set), VS_RFIFO+TS$_p$ stores the cid and set

parameters in the `id` and `set` fields of a variable `start`. When `start` $\neq \perp$, it indicates that VS_RFIFO+TS$_p$ is engaged in a synchronization protocol, during which it exchanges synchronization messages tagged with `start.id` with the end-points in `start.set`; after VS_RFIFO+TS$_p$ delivers a view to its client it resets `start` to $\perp$.

Variable `sync_set` indicates the set of end-points to which a synchronization message tagged with the latest `start.id` has already been sent. When end-point `p` receives `start`$_p$(`cid`, `set`) with a new `cid`, `sync_set` is reset to $\emptyset$ to indicate that a new synchronization message needs to be sent to every end-point in `set`. However, if the `cid` is the same as the last one, `sync_set` is set to `sync_set` $\cap$ `set`. This way, the end-point will send its last synchronization message only to the joining end-points (i.e., those in `set` $-$ `sync_set`), and not to those to which the message was already sent. Notice that the disconnected end-points (i.e., those that are not in `set`) are removed from `sync_set`.

After VS_RFIFO+TS$_p$ receives a `start`$_p$(`cid`, `set`) input from MEMB, it executes an internal action, `set_cut`$_p$(). This action commits `p` to deliver to its client all the messages it has so far received from the members of its current view. For each member `q` of `current_view.set`, `cut`(`q`) is set to the length of the longest continuous prefix of messages in `msgs`[`q`][`current_view`].[4] Action `set_cut`$_p$() results in `p`'s current view being stored in `sync_msg`[`p`][`start.id`].`view`, the committed cut being stored in `sync_msg`[`p`][`start.id`].`cut`, and `sync_set` being set to {`p`}.

VS_RFIFO+TS$_p$ specifies precise preconditions on the CO_RFIFO.`reliable`$_p$(`set`) actions. When VS_RFIFO+TS$_p$ is not engaged in a synchronization protocol (i.e., when `start` $= \perp$), CO_RFIFO is asked to maintain reliable connection just to the end-points in `p`'s current view, `current_view.set`. When VS_RFIFO+TS$_p$ is engaged in a synchronization protocol, it requires CO_RFIFO to maintain reliable connection to the members of a forming view, `start.set`, as well as to those in `current_view.set`. Thus, CO_RFIFO avoids loss of messages sent to the disconnected end-points in case these end-points are later added to the forming view.

After setting the cut and telling CO_RFIFO to maintain reliable connection to everyone in `current_view.set` $\cup$ `start.set`, VS_RFIFO+TS$_p$ uses CO_RFIFO.`send`$_p$ to send the synchronization message `sync_msg`[`p`][`start.id`] tagged with `start.id` to the end-points in `start.set` $-$ `sync_set`, that is, to all those end-points in the proposed membership to which this synchronization message has not already been sent. Afterwards, `sync_set` is adjusted to `start.set`.

When end-point `p` receives synchronization messages from other end-points, via CO_RFIFO.`deliver`$_{q,p}$(⟨'**sync_msg**', `cid`, `v`, `cut`⟩), `p` saves ⟨`v`, `cut`⟩ in `sync_msg`[`q`][`cid`].

VS_RFIFO+TS$_p$ restricts delivery of application messages while it is engaged in a synchronization protocol (i.e., when `start` $\neq \perp$ and `sync_msg`[`p`][`start.id`] $\neq \perp$): Prior to receiving a new view from MEMB, only the messages identified in the cut of its own latest synchronization message, `sync_msg`[`p`][`start.id`].`cut`, can be delivered to the client. After MEMB.`view`$_p$(`v`) occurs, VS_RFIFO+TS$_p$ is allowed to deliver messages identified in the cut `sync_msg`[`q`][`v.startId`(`q`)].`cut` received from `q`, provided `q` is a member of the transitional set from `current_view` to `v`. An end-point `q` $\in$ `current_view.set` $\cap$ `v.set` is considered to be in the transitional set from `current_view` to `v` if `sync_msg`[`q`][`v.startId`(`q`)].`view` is the same as `p`'s `current_view`.

VS_RFIFO+TS$_p$ delivers a view `v` received from MEMB and a transitional set `T` to its client when `p` has received a synchronization message `sync_msg`[`q`][`v.startId`(`q`)]

[4] The longest continuous prefix can be different from the length of `msgs`[`q`][`current_view`] because forwarded messages may arrive out of order and introduce gaps in the `msgs` queues.

from every q in current_view.set ∩ v.set, has computed T, and has delivered all the application messages identified in the cuts of the members of T, as specified by the last three preconditions on $\texttt{view}_p(v, T)$. The first two preconditions ensure, respectively, that no new MEMB.$\texttt{start}_p$ notification was issued after MEMB.$\texttt{view}_p(v)$ and that p has sent its synchronization message to everybody in v.set. The third precondition specifies that p has sent to others all of its own messages indicated in its own cut. All these preconditions work in conjunction with those in WV_RFIFO.$\texttt{view}_p(v)$.

Recall from section 6.1 that WV_RFIFO_p allows for nondeterministic forwarding of other end-points' application mesages. VS_RFIFO+TS_p resolves this nondeterminism by placing two additional preconditions on CO_RFIFO.$\texttt{send}_p(\texttt{set}, \langle \text{'}\textbf{fwd\_msg}\text{'}, r, v, m, i\rangle)$: The first checks a variable forwarded_set to make sure that message m was not previously forwarded to anyone in set. The second tests that a certain predicate, called ForwardingStrategyPredicate(set, r, v, i), holds. This predicate is designed to ensure that all end-points in the transitional set T are able to deliver all the messages that each has committed to deliver in its synchronization message, in particular those sent by disconnected clients. End-points test ForwardingStrategyPredicate to decide whether they need to forward any messages to others.

6.2.1. Forwarding strategy predicate. We now provide two examples of ForwardingStrategyPredicates. With the first, multiple copies of the same message may be forwarded by different end-points. The second strategy reduces the number of forwarded copies of a message. Many other possible strategies exist. For example, a strategy can employ randomization to decide whether an end-point should forward a message in a certain time slice, and suppress forwarding of messages that have already been forwarded by others.

A simple strategy. With our first strategy, end-point p forwards message m only if p has committed to deliver m. In addition, if m was originally sent in view v, p forwards m to an end-point q only if p does not know of any view of q later than v and if the latest sync_msg from q sent in view v indicates that q has not received message m.

```
ForwardingStrategyPredicate(set, r, v, i) ≡
 (∃ cid) (sync_msg[p][cid].view = v  ∧  i ≤ sync_msg[p][cid].cut(r))
 ∧ set = { q |  view_msg[q] ≤ v  ∧  (∃ cid′) (sync_msg[q][cid′].view = v
               ∧ (∄ cid′′ > cid′) sync_msg[q][cid′′].view = v
               ∧ sync_msg[q][cid′].cut(r) < i) }.
```

If some end-point q is missing a certain message m, then m will be forwarded to q by some end-point p that has committed to deliver m, when p learns from q's synchronization message that q misses m.

Reducing the number of forwarded copies of a message. The second strategy relies on the computed transitional set T from view v to v′ to decide which message should be forwarded by which member of the transitional set. Assume that a member u of T misses a message m that was originally sent in v by a nonmember r of T, but that was committed to delivery by some other members of T. Among these members, ForwardingStrategyPredicate selects the one with the minimal process identifier to forward m to u; variations of this predicate may use a different deterministic rule for selecting a member, for example, accounting for network topology or communication costs. The selected end-point, p, forwards the message to u only if view v′ is the latest view known to p, as specified by the first conjunct below. Otherwise, v′ is an obsolete view, so there is no need to help u transition in to v′. The described strategy does not forward to $u \in T$ messages from the members of T because

u is guaranteed to receive these messages directly from their original senders (unless v′ becomes obsolete because further view changes occur).

```
ForwardingStrategyPredicate(set, r, v, i) ≡
  Let v′ = memb_view ∧                          // latest view known to {p}
  sync_msg[p][v′.startId(p)] ≠ ⊥ ∧               // already sent own sync_msg
  Let v = sync_msg[p][v′.startId(p)].view ∧
  (for all q ∈  v.set ∩ v′.set) sync_msg[q][v′.startId(q)] ≠ ⊥  ∧  // received right sync_msgs
  Let T = {q ∈ v.set ∩ v′.set | sync_msg[q][v′.startId(q)].view = v} ∧
  r ∉ T   ∧                              // only forward messages from end-point not in T
  set = {u ∈ T  | sync_msg[u][v′.startId(u)].cut(r) < i }  ∧
  p = min{u ∈ T  | sync_msg[u][v′.startId(u)].cut(r) ≥ i }.
```

If all end-points receive the same view from MEMB, only one copy of m will be forwarded to each u. In rare cases, however, when MEMB delivers different views to different end-points, more than one end-point may forward the same message m to the same end-point u.

Each end-point waits to receive a new view from MEMB and all the right synchronization messages before it forwards messages to others. Thus, compared to the first strategy, this strategy reduces the communication traffic at the cost of slower recovery of lost messages.

6.2.2. Correctness argument. The VS_RFIFO+TS automaton, resulting from the composition of all end-point automata and the MEMB and CO_RFIFO automata, satisfies the VSRFIFO : SPEC and TS : SPEC safety specifications, as well as liveness (Property 5.2), as we formally prove in Appendixes B.2, B.3, and C, respectively. Below we give highlights of these proofs.

VSRFIFO : SPEC is a modification of WV_RFIFO : SPEC. The proof that VS_RFIFO+TS satisfies VSRFIFO : SPEC reuses the proof that WV_RFIFO satisfies WV_RFIFO : SPEC and involves reasoning about only how VSRFIFO : SPEC modifies WV_RFIFO : SPEC. The proof extends refinement mapping R between WV_RFIFO and WV_RFIFO : SPEC with a mapping R_n. R_n maps the cut used by the end-points of VS_RFIFO+TS to move from a view v to a view v′ to the cut[v][v′] variable of VSRFIFO : SPEC. The proof depends on Invariant B.9 and Corollary B.1, which state that all end-points that move from a view v to a view v′ use the same synchronization messages, compute the same transitional set T, and therefore use the same cut.

The proof in Appendix B.3 shows that VS_RFIFO+TS satisfies TS : SPEC. The proof augments VS_RFIFO+TS_p with a *prophecy variable* that guesses, at the time end-point p receives a $\texttt{start}_p(\texttt{cid}, \texttt{set})$ notification from MEMB, possible future views that may contain cid in their startId(p) mappings. For each of these views v′, VS_RFIFO+TS simulates a $\texttt{set\_prev\_view}_p(v')$ action of TS : SPEC, thereby fixing the previous view of v′ to be p's current view v.

In a fair execution of VS_RFIFO+TS in which the same last view v′ is delivered to all its members and no start events subsequently occur, the three preconditions on the $\texttt{view}_p(v', T_p)$ delivery are eventually satisfied for every p ∈ v′.set:

1. Condition v′.startId(p) = start.id remains true since the execution has no subsequent start events at p.

2. End-point p eventually receives synchronization messages tagged with the "right" cid from every member of v.set ∩ v′.set because they keep taking steps towards reliably sending these synchronization messages to p (by low-level fairness of the code) and because CO_RFIFO eventually delivers these messages to p (by the liveness assumption on CO_RFIFO).

3. End-point p eventually receives and delivers all the messages committed to

AUTOMATON GCS_p = VS_RFIFO+TS+SD$_p$ MODIFIES VS_RFIFO+TS$_p$

Signature Extension:

Input: block_ok$_p$() new

Internal: set_cut$_p$() modifies set_cut$_p$()

Output: block$_p$() new

view$_p$(v,T) modifies vs_rfifo+ts.view$_p$(v,T)

State Extension:

block_status ∈ {unblocked, requested, blocked}, initially unblocked

Transition Restriction:

INTERNAL **set_cut$_p$**()
pre: block_status = blocked

OUTPUT **block$_p$**()
pre: start ≠ ⊥
block_status = unblocked
eff: block_status ← requested

OUTPUT **view$_p$**(v,T)
eff: block_status ← unblocked

INPUT **block_ok$_p$**()
eff: block_status ← blocked

FIG. 6.5. GCS_p *end-point automaton.*

in the cuts of the members of the transitional set T_p because for each such message there is at least one end-point in T_p that has the message in its msgs buffer and that will reliably forward it to p (according to the ForwardingStrategyPredicate) if necessary. Also, p never delivers any messages beyond those committed to in the cuts of the members of T_p because of the precondition on application message delivery.

6.3. Adding support for self-delivery. As a final step in constructing the automaton that models an end-point of our group communication service, GCS_p, we add support for self-delivery to the VS_RFIFO+TS$_p$ automaton presented above. Self-delivery requires each end-point to deliver to its client all the messages the client sends in a view, before moving on to the next view.

In order to implement self-delivery, virtually synchronous delivery, and within-view delivery together in a live manner, each end-point must *block* its client from sending new messages while a view change is taking place (as proven in [23]). Therefore, we add to VS_RFIFO+TS$_p$ an output action block and an input action block_ok. We assume that the client at end-point p has the matching actions and that it eventually responds to every block request with a block_ok response and subsequently refrains from sending messages until a view is delivered to it. In section B.4, we formalize this requirement as an abstract client automaton.

The GCS_p automaton appears in Figure 6.5. After receiving the first start notification in a given view, end-point p issues a block request to its client and awaits receiving a block_ok response before executing set_cut$_p$(). As a result of set_cut$_p$(), p commits to deliver all the messages its client has sent in the current view. Therefore, p has to deliver all these messages before moving on to a new view, and self-delivery is satisfied. Due to the use of inheritance, the GCS automaton preserves all the safety properties satisfied by its parent. Since end-point p has its own messages on the msgs[p][p] queue and can deliver them to its client, liveness is also preserved. Thus, GCS satisfies all the properties we have specified in section 5.

6.4. Optimizations and extensions. Having formally presented the basic algorithm for an end-point of our virtually synchronous GCS, we now discuss several optimizations and extensions that can be added to the algorithm to make its implementation more practical. Specifically, we discuss ways to reduce the size and number of synchronization messages. (In general, the number of synchronization messages and the size of each message sent during a synchronization protocol can be linear in the number of members.) We also discuss garbage collection and ways to avoid the use of nonvolatile storage.

The first optimization that reduces the size of synchronization messages relies on the following observation: An end-point p does not need to send its current view and its cut to end-points that are not in current_view.set because p cannot be included in their transitional sets. However, these end-points still need to hear from p if p is in their current views. Therefore, end-point p could send a smaller synchronization message to the end-points in start.set − current_view.set, containing its start.id only (but neither a view nor a cut). This message would be interpreted as saying "I am not in your transitional set," and the recipients of this message would know not to include p in their transitional sets for views v' with v'.startId(p) = p's start.id. When using this optimization, p also does not need to include its current view in the synchronization messages sent to current_view.set − start.set, since the view information can be deduced from p's view_msg.

An additional optimization can be used if we strengthen the membership specification to require a MEMB.start with a new identifier to be sent every time MEMB changes its mind about the membership of a forming view. In this case, the latest MEMB.start has the same membership as the delivered MEMB.view. Therefore, the synchronization messages can be shortened to not include information about application messages delivered from end-points in start.set ∩ current_view.set: for an end-point p, end-points that have p in their transitional sets will deliver all the application messages that p sent before its synchronization message.

Other optimizations can reduce the total number of messages sent during synchronization protocol by all end-points. A simple way to do this is to transform the algorithm into a leader-based one, as in [44, 40]. A more scalable approach was suggested by Guo, Vogels, and van Renesse [26]. Their algorithm uses a two-level hierarchy for message dissemination in order to implement VS: end-points send synchronization messages to their designated leaders, which in turn exchange only the cumulative information among themselves. Also, the number of messages exchanged to synchronize multiple groups can be reduced, as suggested in [11, 39], by aggregating information pertaining to multiple groups into a single message.

Another optimization addresses the use of stable storage. Recall that in section 4 we assumed that end-points keep their running states on stable storage, and therefore, recover with their state intact. However, our group multicast service does provide meaningful semantics even when GCS end-points maintain their running state on volatile storage. When an end-point p recovers after a crash, it can start executing with its state reset to an initial value with current_view being the singleton view v_p. It needs to contact the MEMB service to be readmitted to its groups. The client would refrain from sending any messages in its recovered view until it receives a new view from its end-point. This view would satisfy local monotonicity and self-inclusion because these are the properties guaranteed by the MEMB service. The specification of virtually synchronous delivery should be changed so that recovery is interpreted as delivering a singleton view. The remaining safety properties are also preserved because they involve message delivery within a single view.

In a practical implementation of our service, some sort of garbage collection mechanism is required in order to keep the buffer sizes finite. The implementation of [43] discards messages from older views when moving to a new view and also when learning that they were already delivered to every client in the view. This implementation also discards older synchronization messages: an end-point holds on to only the latest synchronization message it has received from each end-point. This optimization does not violate liveness since discarded synchronization messages necessarily pertain to

obsolete views.

7. Conclusions. We have designed a novel group multicast service targeted for WANs. Our service implements a variant of the VS semantics that includes a collection of properties that have been shown useful for many distributed applications (see [16]). Many GCSs, for example, [44, 40, 9, 5, 19], support these and similar properties. Our design has been implemented [43] (in C++) as part of a novel architecture for scalable group communication in WANs using the datagram service of [7] and the Moshe membership algorithm [31].

The main contribution of this paper is a VS algorithm run by GCS end-points, in particular, its synchronization protocol, which enforces virtually synchronous delivery. This protocol is invoked when the underlying membership service begins to form a new view, and is run while the view is forming. The protocol involves a single message-exchange round during which members of the forming view send synchronization messages to each other. In contrast to previously suggested VS algorithms (e.g., [23, 5, 26, 3, 9]), our algorithm does not require end-points to preagree upon a globally unique identifier before sending synchronization messages and thus involves less communication. Performing less communication is especially important in WANs, where message latency tends to be high.

Furthermore, unlike the algorithms in [5, 26, 9, 40], our algorithm allows the membership service to change the membership of a forming view while the synchronization protocol is running; the protocol responds immediately to such membership changes.

We are not aware of any other algorithm for VS that does not preagree on common identifiers before sending synchronization messages and that always allows new members to join a forming view while the synchronization protocol is running. Our algorithm achieves these two features by virtue of a simple yet powerful idea: End-points tag their synchronization messages with start identifiers that are locally generated by the membership service; when the membership service forms a view and delivers it to the end-points, the view includes information about which start identifiers were given to which member. This information communicates to the end-points which synchronization messages they need to consider from each member. Since no preagreement upon a common identifier takes place, there is nothing that would inhibit the membership service and the VS algorithm from allowing new members to join the forming view; end-points just have to forward their last synchronization messages to the joiners.

As a second contribution of this paper, our design has demonstrated how to effectively decouple the algorithm for achieving VS from the algorithm for maintaining membership. As argued in [6, 31], such decoupling is important for providing efficient and scalable group communication services in WANs. In previous designs that implement VS atop an external membership service [11, 40], the membership service is not allowed to add new members to an already forming view, and the membership service waits to synchronize with all end-points of the formed view before delivering the view to any of the clients.

A distinct and important characteristic of our design is the high level of formality and rigor at which it has been carried out. This paper has provided precise descriptions of the GCS algorithm and the semantics it provides, as well as a formal proof of the algorithm's correctness—both safety and liveness. Previously, formal approaches based on I/O automata were used to specify the semantics of VS GCSs and to model and verify their applications, for example, in [15, 22, 18, 33, 27]. However,

due to their size and complexity, VS algorithms were not previously modeled using formal methods, nor were they assertionally verified. Our experience has taught us the importance of careful modeling and verification: in the process of proving our algorithm's correctness we have often uncovered subtleties and ambiguities that had to be resolved.

In order to manage the complexity of our design, we developed a new formal inheritance-based methodology [30]. The incremental way in which we constructed our algorithms and specifications also allowed us to construct the simulation proof incrementally. For example, in order to prove that VS_RFIFO+TS simulates VS_RFIFO+TS : SPEC, we extended the simulation relation from WV_RFIFO to WV_RFIFO : SPEC and reasoned solely about the extension, without repeating the reasoning about the parent components (see Appendix B.2). This reuse was justified by the proof extension theorem of [30] (see Appendix A.3). The use of incremental construction was the key to our success in formally modeling and reasoning about such a complex and sophisticated service. We hope that the methodology employed in this paper shall also be helpful to other researchers working on formal modeling of complex distributed systems.

Appendix A. Review of proof techniques. In this section we describe the main techniques used to prove correctness of I/O automata: invariant assertions, hierarchical proofs, refinement mappings, and history and prophecy variables. The material in this section is closely based on [36, pp. 216–228] and [35, pp. 3, 4, and 13]. In section A.3 we present a proof extension theorem of [30] that provides a formal framework for the reuse of simulation proofs based on refinement mappings.

A.1. Invariants. The most fundamental type of property to be proved about an automaton is an *invariant assertion*, or just *invariant* for short. An invariant assertion of an automaton A is defined as any property that is true in every single reachable state of A.

Invariants are typically proved by induction on the number of steps in an execution leading to the state in question. While proving an inductive step, we consider only *critical actions*, which affect the state variables appearing in the invariant.

A.2. Hierarchical proofs. One of the important proof strategies is based on a hierarchy of automata. This hierarchy represents a series of descriptions of a system or algorithm at different levels of abstraction. The process of moving through the series of abstractions, from the highest level to the lowest level, is known as *successive refinement*. The top level may be nothing more than a problem specification written in the form of an automaton. The next level is typically a very abstract representation of the system: it may be centralized rather than distributed, or have actions with large granularity, or have simple but inefficient data structures. Lower levels in the hierarchy look more and more like the actual system or algorithm that will be used in practice: they may be more distributed, have actions with small granularity, and contain optimizations. Because of all this extra detail, lower levels in the hierarchy are usually harder to understand than the higher levels. The best way to prove properties of the lower-level automata is by relating these automata to automata at higher levels in the hierarchy, rather than by carrying out direct proofs from scratch.

A.2.1. Refinement mappings. The simplest way to relate two automata, say A and S, is to present a *refinement mapping* R from the reachable states of A to the reachable state of S such that it satisfies the following two conditions:

1. If t_0 is an initial state of A, then $R(s_0)$ is an initial state of S.

2. If t and $R(t)$ are reachable states of A and S, respectively, and (t, π, t') is a step of A, then there exists an execution fragment of S beginning at state $R(t)$ and ending at state $R(t)'$, with its trace being the same as the trace of π and its final state $R(t)'$ being the same as $R(t')$.

The first condition asserts that any initial state of A has some corresponding initial state of S. The second condition asserts that any step of A has a corresponding sequence of steps of S. This corresponding sequence can consist of one step, many steps, or even no steps, as long as the correspondence between the states is preserved and the external behavior is the same.

The following theorem gives the key property of refinement mappings.

THEOREM A.1. *If there is a refinement mapping from A to S, then* `traces`*(A)* $\subseteq$ `traces`*(S).*

If automata A and S have the same external signature and the traces of A are the traces of S, then we say that A *implements* S *in the sense of trace inclusion*, which means that A never does anything that S couldn't do. Theorem A.1 implies that, in order to prove that one automaton implements another in the sense of trace inclusion, it is enough to produce a refinement mapping from the former to the latter.

A.2.2. History and prophecy variables. Sometimes, however, even when the traces of A are the traces of S, it is not possible to give a refinement mapping from A to S. This may happen due to the following two generic reasons:

1. The states of S may contain more information than the states of A.
2. S may make some premature choices, which A makes later.

The situation when A has been optimized not to retain certain information that S maintains can be resolved by augmenting the state of A with additional components, called *history variables* (because they keep track of additional information about the history of execution), subject to the following constraints:

1. Every initial state has at least one value for the history variables.
2. No existing step is disabled by the addition of predicates involving history variables.
3. A value assigned to an existing state component must not depend on the value of a history variable.

These constraints guarantee that the history variables simply record additional state information and do not otherwise affect the behavior exhibited by the automaton. If the automaton A_{HV} augmented with history variables can be shown to implement S by presenting a refinement mapping, it follows that the original automaton A without the history variables also implements S because they have the same traces.

The situation when S is making a premature choice, which A makes later, can be resolved by augmenting A with a different sort of auxiliary variable, *prophecy variable*, which can look into the future just as history variable looks into the past. A prophecy variable guesses in advance some nondeterministic choice that A is going to make later. The guess gives enough information to construct a refinement mapping to S (which is making the premature choice). For an added variable to be a prophecy variable, it must satisfy the following conditions:

1. Every state has at least one value for the prophecy variable.
2. No existing step is disabled *in the backward direction* by the new preconditions involving a prophecy variable. More precisely, for each step (t, π, t') there must be a state (t, p) and a p such that there is a step $((t, p), \pi, (t', p'))$.
3. A value assigned to an existing state component must not depend on the value of the prophecy variable.

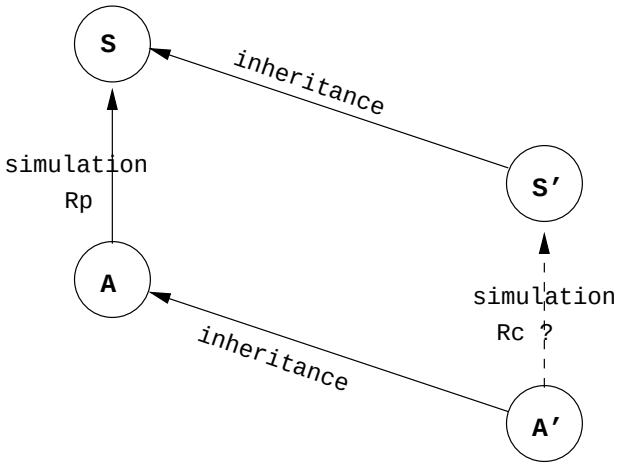


FIG. A.1. *Algorithm* A *simulates specification* S *with* R. *Can* R *be reused for building a refinement* R′ *from a child* A′ *of* A *to a child* S′ *of* S*?*

4. If t is an initial state of A and (t, p) is a state of the A augmented with the prophecy variable, then it must be its initial state.

If these conditions are satisfied, the automaton augmented with the prophecy variable will have the same (finite) traces as the automaton without it. Therefore, if we can exhibit a refinement mapping from A_{PV} to S, we know that the A implements S.

A.3. Inheritance and proof extension theorem. We now present a theorem from [30] which lays the foundation for incremental proof construction. Consider the example illustrated in Figure A.1, where a refinement mapping R from an algorithm A to a specification S is given, and we want to construct a refinement mapping R′ from a child A′ of an automaton A to a child S′ of a specification automaton S.

Theorem A.2 below implies that such a refinement R′ can be constructed by supplementing R with a mapping $\mathtt{R_n}$ from the states of A′ to the state extension introduced by S′. Mapping $\mathtt{R_n}$ has to map every initial state of A′ to some initial state extension of A′ and it has to satisfy a step condition similar to the one for refinement mapping (section A.2.1), but only involving the transition restriction of S′.

THEOREM A.2. *Let automaton* A′ *be a child of automaton* A. *Let automaton* S′ *be a child of automaton* S. *Let mapping* R *be a refinement from* A *to* S.

Let $\mathtt{R_n}$ *be a mapping from the states of* A′ *to the state extension introduced by* S′.

A mapping R′ *from the states of* A′ *to the states of* S′, *defined in terms of* R *and* $\mathtt{R_n}$ *as*

$$\mathtt{R'}(\langle \mathtt{t}, \mathtt{t_n} \rangle) = \langle \mathtt{R}(\mathtt{t}), \mathtt{R_n}(\langle \mathtt{t}, \mathtt{t_n} \rangle) \rangle,$$

is a refinement *from* A′ *to* S′ *if* R′ *satisfies the following two conditions:*

1. *If* t *is an initial state of* A′, *then* $\mathtt{R_n}(\mathtt{t})$ *is an initial state extension of* S′.
2. *If* $\langle \mathtt{t}, \mathtt{t_n} \rangle$ *is a reachable state of* A′, $\mathtt{s} = \langle \mathtt{R}(\mathtt{t}), \mathtt{R_n}(\langle \mathtt{t}, \mathtt{t_n} \rangle) \rangle$ *is a reachable state of* S′, *and* $(\langle \mathtt{t}, \mathtt{t_n} \rangle, \pi, \langle \mathtt{t}', \mathtt{t}'_\mathtt{n} \rangle)$ *is a step of* A′, *then there exists a finite sequence* α *of alternating states and actions of* S′, *beginning from* s *and ending at some state* s′, *satisfying the following conditions:*

1. α *projected onto states of* S *is an execution sequence of* S.

2. *Every step* $(\mathtt{s}_i, \sigma, \mathtt{s}_{i+1}) \in \alpha$ *is consistent with the transition restriction placed on* S *by* S′.
3. *The parent component of the final state* s′ *is* $\mathtt{R}(\mathtt{t}')$.
4. *The child component of the final state* s′ *is* $\mathtt{R}_n(\langle \mathtt{t}', \mathtt{t}'_n \rangle)$.
5. α *has the same trace as* π.

In practice, one would exploit this theorem as follows: The simulation proof between the parent automata already provides a corresponding execution sequence of the parent specification for every step of the parent algorithm. It is typically the case that the same execution sequence, padded with new state variables, corresponds to the same step at the child algorithm. Thus, conditions 1, 3, and 5 of Theorem A.2 hold for this sequence. The only conditions that have to be checked are 2, and 4, that is, that every step of this execution sequence is consistent with the transition restriction placed on S by S′ and that the values of the new state variables of S′ in the final state of this execution match those obtained when $\mathtt{R}_n$ is applied to the poststate of the child algorithm.

A.4. Safety versus liveness. Proving that one automaton implements another in the sense of trace inclusion constitutes only *partial correctness*, as it implies *safety* but not *liveness*. In other words, partial correctness ensures than "bad" things never happen, but it does not say anything about whether some "good" thing eventually happens.

In this paper, we use invariant assertions and simulation techniques to prove that our algorithms satisfy safety properties, which are stated as I/O automata. For liveness proofs, we use a combination of invariant assertions and carefully proven operational arguments.

Appendix B. Correctness proof: Safety properties. We now formally prove, using invariant assertions and simulations, that our algorithms satisfy the safety properties of section 5.1. Proofs done with invariant assertions and simulations are verifiable (even by a computer) because they involve reasoning only about single steps of the algorithm. A review of the proof techniques used in this section appears in Appendix A.

The safety proof is *modular*: we exploit the inheritance-based structure of our specifications and algorithms to reuse proofs. In section B.1 we prove correctness of the within-view reliable FIFO multicast service by showing a refinement mapping from WV_RFIFO to WV_RFIFO : SPEC. In section B.2 we extend this refinement mapping to map the new state added in VS_RFIFO+TS to that in VSRFIFO : SPEC. In section B.3 we prove that VS_RFIFO+TS also simulates TS : SPEC. Finally, in section B.4 we extend the refinement above to map the new state of GCS to that of SELF : SPEC. The proof extension theorem of [30] (also reviewed in Appendix A) implies that the GCS automaton satisfies WV_RFIFO : SPEC, VSRFIFO : SPEC, TS : SPEC, and SELF : SPEC.

B.1. Within-view reliable FIFO multicast. Intuitively, in order to simulate WV_RFIFO : SPEC with WV_RFIFO, we need to show that WV_RFIFO satisfies self-inclusion and local monotonicity for delivered views, and we need to show that the ith message delivered by q from p in view v is the ith message sent in view v by the client at p. In order to prove this, we need to show that the algorithm correctly associates messages with the views in which they were sent and with their indices in the sequences of messages sent in these views. We split the proof into three parts: section B.1.1 states key invariants but defers the proof of one of them to section B.1.3; section B.1.2 contains the simulation proof.

B.1.1. Key invariants. The following invariant captures the self-inclusion property.

INVARIANT B.1 (self-inclusion). *In every reachable state* `s` *of* WV_RFIFO*, for all* `Proc p`, $\texttt{p} \in \texttt{s[p].memb\_view.set}$ *and* $\texttt{p} \in \texttt{s[p].current\_view.set}$.

Proof of Invariant B.1. The proof immediately follows from the MEMB specification. □

The local monotonicity property follows directly from the precondition, $\texttt{v.id} > \texttt{memb\_view}$, of the MEMB.$\texttt{view}_\texttt{p}(\texttt{v})$ actions.

The following invariant relates application messages at different end-points' queues to the corresponding messages on the original senders' queues.

INVARIANT B.2 (message consistency). *In every reachable state* `s` *of* WV_RFIFO*, for all* `Proc p` *and* `Proc q`*, if* $\texttt{s[q].msgs[p][v][i]} = \texttt{m}$*, then* $\texttt{s[p].msgs[p][v][i]} = \texttt{m}$.

This proposition is vacuously true in the initial state because all message queues are empty. For the inductive step, we have to consider the CO_RFIFO.$\texttt{deliver}_{\texttt{q,p}}(\langle \text{`}\textbf{app\_msg}\text{'}, \texttt{m}\rangle)$ and CO_RFIFO.$\texttt{deliver}_{\texttt{q,p}}(\langle \text{`}\textbf{fwd\_msg}\text{'}, \texttt{r}, \texttt{v}, \texttt{m}, \texttt{i}\rangle)$ actions and have to argue that the message `m` that these actions deliver is placed in the right place in `q`'s `msgs` buffer. The proof of this invariant appears in section B.1.3 after the simulation proof.

B.1.2. Simulation.

LEMMA B.1. *The following function* `R` *is a* refinement mapping *from automaton* WV_RFIFO *to automaton* WV_RFIFO : SPEC *with respect to their reachable states.*

```
R(s∈ReachableStates(WV_RFIFO)) = t∈ReachableStates(WV_RFIFO : SPEC), where

  For each Proc p, View v:       t.msgs[p][v]   =   s[p].msgs[p][v]

  For each Proc p, Proc q: t.last_dlvrd[p][q]   =   s[q].last_dlvrd[p]

  For each Proc p:          t.current_view[p]   =   s[p].current_view
```

Proof of Lemma B.1.

Action correspondence. Automaton WV_RFIFO : SPEC has three types of actions. Actions $\texttt{view}_\texttt{p}(\texttt{v})$, $\texttt{send}_\texttt{p}(\texttt{m})$, and $\texttt{deliver}_\texttt{p}(\texttt{q}, \texttt{m})$ are simulated when WV_RFIFO takes the corresponding $\texttt{view}_\texttt{p}(\texttt{v})$, $\texttt{send}_\texttt{p}(\texttt{m})$, and $\texttt{deliver}_\texttt{p}(\texttt{q}, \texttt{m})$ actions. Steps of WV_RFIFO involving other actions correspond to empty steps of WV_RFIFO : SPEC.

Simulation proof. For the most part, the simulation proof is straightforward. Here, we present only the interesting steps.

The fact that the corresponding step of WV_RFIFO : SPEC is enabled when WV_RFIFO takes a step involving $\texttt{view}_\texttt{p}(\texttt{v})$ relies on $\texttt{p} \in \texttt{memb\_view.set}$ (Invariant B.1).

For the steps involving the $\texttt{deliver}_\texttt{p}(\texttt{q}, \texttt{m})$ action, in order to deduce that the corresponding step of WV_RFIFO : SPEC is enabled, we need to know that the message located at index $\texttt{s[p].last\_dlvrd[q]} + 1$ on the $\texttt{s[p].msgs[q][s[p].current\_view]}$ queue is the same message that end-point `q` has on its corresponding queue at the same index. This property is implied by Invariant B.2.

Steps that involve receiving original and forwarded application messages from the network simulate empty steps of WV_RFIFO : SPEC. Among these steps the only critical ones are those that deliver a message from `p` to `p` because they may affect $\texttt{s[p].msgs[p][p]}$ queue. Since end-points do not send messages to themselves, such steps may not happen. Indeed, action CO_RFIFO.$\texttt{send}_\texttt{p}(\texttt{set}, \langle \text{`}\textbf{app\_msg}\text{'}, \texttt{m}\rangle)$ has a precondition $\texttt{set} = \texttt{s[p].current\_view.set} - \{\texttt{p}\}$, and action CO_RFIFO.$\texttt{send}_\texttt{p}(\texttt{set}, \langle \text{`}\textbf{fwd\_msg}\text{'}, \texttt{r}, \texttt{v}, \texttt{m}, \texttt{i}\rangle)$ has a precondition $\texttt{p} \notin \texttt{set}$. □

From Lemma B.1 and Theorem A.1 we conclude the following.

THEOREM B.1. WV_RFIFO *implements* WV_RFIFO : SPEC *in the sense of trace inclusion.*

B.1.3. Auxiliary invariants. We now state and prove a number of auxiliary invariants necessary for the proof of the key message consistency invariant (Invariant B.2).

In any view, before an end-point sends a `view_msg` to others (and hence before it sends any application message to others), it tells CO_RFIFO to maintain reliable connection to every member of its current view. The following invariant captures this property.

INVARIANT B.3 (connection reliability). *In every reachable state* `s` *of* WV_RFIFO*, for all* `Proc p`*, if* `s[p].current_view = s[p].view_msg[p]`*, then* `s[p].current_view.set` $\subseteq$ `s[p].reliable_set`*.*

Proof of Invariant B.3. By induction on the length of the execution sequence, the proof follows directly from the code. □

After an end-point delivers a new view to its client, it sends a `view_msg` to other members of the view. The stream of `view_msg`s that an end-point sends to others is monotonic because the delivered views satisfy local monotonicity. The following invariant captures this property. It states that the subsequence of messages in transit from end-point `p` to end-point `q` solely consisting of the `view_msg`s is monotonically increasing. It also relates the current view of an end-point `p` to the view contained in the `p`'s latest `view_msg` to `q`.

INVARIANT B.4 (monotonicity of view messages). *Let* `s` *be a reachable state of* WV_RFIFO*. Consider the subsequence of messages in* `s.channel[p][q]` *of the* `ViewMsg` *type. Examine the sequence of views included in these view messages, and construct a new sequence* seq *of views by prepending this view sequence with the element* `s[q].view_msg[p]`*. For all* `Proc p`, `Proc q`*, the following propositions are true:*

1. *The sequence* seq *is (strictly) monotonically increasing.*
2. *If* `s[p].current_view` $\neq$ `s[p].view_msg[p]`*, then* `s[p].current_view` *is strictly greater then the last (largest) element of* seq*.*
3. *If* `s[p].current_view = s[p].view_msg[p]`*, and if* `q` $\in$ `s[p].current_view.set`*, then* `s[p].current_view` *is equal to the last (largest) element of* seq*.*

Proof of Invariant B.4. All three propositions are true in the initial state. We now consider steps involving the critical actions.

CO_RFIFO.`lose(p, q)`. The first two propositions remain true because this action throws away only the last message from the CO_RFIFO `s.channel[p][q]`. The third proposition is vacuously true because `q` $\notin$ `s[p].current_view.set`. If it were, the CO_RFIFO.`lose(p, q)` action would not be enabled because Invariant B.3 would imply that `s[p].current_view.set` is a subset of `s[p].reliable_set`, which would then imply that `q` $\in$ `s.reliable_set[p]` (because `s[p].reliable_set` = `s.reliable_set[p]`, as can be shown by straightforward induction).

$\texttt{view}_{\texttt{p}}$`(v)`. The first proposition is unaffected. The second proposition follows from the inductive hypothesis and the precondition `v.id` $>$ `s[p].current_view.id`. The third proposition is vacuously true because `s[p].current_view` $\neq$ `s[p].view_msg[p]` as follows from the precondition `v.id` $>$ `s[p].current_view.id` and the fact that, in every reachable state `s`, `s[p].current_view` $\geq$ `s[p].view_msg[p]` (as can be proved by straightforward induction).

CO_RFIFO.$\texttt{send}_{\texttt{p}}$`(set,` $\langle$`'view_msg', v`$\rangle$`)`. The first proposition is true in the poststate because of the inductive hypothesis of the second proposition. The second proposition is vacuously true in the poststate. The third proposition is true in the

poststate because of the effect of this action.

CO_RFIFO.$\texttt{deliver}_{\texttt{p,q}}(\langle$'**view_msg**', v$\rangle$). It is straightforward to see that all three propositions remain true in the poststate. □

History tags. In order to reason about original application messages traveling on CO_RFIFO channels, we need a way to reference, for each of these messages, the view in which it was originally sent and its index in the FIFO ordered sequence of messages sent in that view. To this end, we augment each original application message $\langle$'**app_msg**', m$\rangle$ with two *history tags*, Hv and Hi, that are set to current_view and last_sent + 1, respectively, when CO_RFIFO.$\texttt{send}_{\texttt{p}}(\texttt{set}, \langle$'**app_msg**', m$\rangle$) occurs. (See Appendix A for details on history variables.)

```
OUTPUT co_rfifo.send_p(set, ⟨'app_msg', m, Hv, Hi⟩)
pre: ...
     Hv = current_view
     Hi = last_sent + 1
 eff: ...
```

With the history tags, the interface between WV_RFIFO and CO_RFIFO for handling original application messages becomes CO_RFIFO.$\texttt{send}_{\texttt{p}}(\texttt{set}, \langle$'**app_msg**', m, Hv, Hi$\rangle$) and CO_RFIFO.$\texttt{deliver}_{\texttt{p,q}}(\langle$'**app_msg**', m, Hv, Hi$\rangle$).

The goal of the next three invariants is to show that, when end-point q receives an application message m tagged with a history view Hv and a history index Hi, the current value of q's view_msg[p] equals Hv and that of last_rcvd[p] + 1 equals Hi.

INVARIANT B.5 (history view consistency). *In every reachable state* s *of* WV_RFIFO*, for all* Procp, Procq*, the following holds. For all messages* $\langle$'**app_msg**', m, Hv, Hi$\rangle$ *on the* CO_RFIFO s.channel[p][q]*, view* Hv *equals either the view of the closest preceding view message on* s.channel[p][q] *if there is such, or* s[q].view_msg[p] *otherwise.*

Proof of Invariant B.5. The proof follows by induction. A step involving CO_RFIFO.$\texttt{send}_{\texttt{p}}(\texttt{set}, \langle$'**app_msg**', m, Hv, Hi$\rangle$) directly follows from part 3 of Invariant B.4. The proposition is not affected by steps involving CO_RFIFO.lose(p, q) because those may only remove the last messages from the CO_RFIFO s.channel[p][q]. The other steps are straightforward. □

The following invariant states that the value of s[p].last_sent equals the number of application messages that p sent in its current view and that are either still in transit on the CO_RFIFO s.channel[p][q] or are already received by q.

INVARIANT B.6. *In every reachable state* s *of* WV_RFIFO*, for all* Procp *and for all* Procq $\in$ s[p].current_view.set $-$ {p}*, the following is true.*

$$\begin{aligned}
&\texttt{s[p].last_sent} \\
&= \left|\{\texttt{msg} \in \texttt{s.channel[p][q]} \ : \ \texttt{msg} \in \texttt{AppMsg and msg.Hv} = \texttt{s[p].current_view}\}\right| \\
&+ \begin{cases} \texttt{s[q].last_rcvd[p]} & \textit{if } \texttt{s[q].view_msg[p]} = \texttt{s[p].current_view}, \\ 0 & \textit{otherwise.} \end{cases}
\end{aligned}$$

Proof of Invariant B.6. The proof follows by induction. Consider steps involving the following critical actions.

CO_RFIFO.lose(p, q). Assume that the last message on queue s.channel[p][q] is an application message msg with msg.Hv = s[p].current_view. If a step involving CO_RFIFO.lose(p, q) action could occur, then the proposition would be false. However, as we are going to argue now, q $\in$ s.reliable_set[p], so such a step cannot occur.

We can prove by induction that msg $\in$ s.channel[p][q] implies s[p].view_msg[p] = s[p].current_view. By Invariant B.3, s[p].current_view.set $\subseteq$ s[p].reliable_set.

Since $\mathtt{q} \in \mathtt{s}[\mathtt{p}].\mathtt{current\_view.set}$ and $\mathtt{s}[\mathtt{p}].\mathtt{reliable\_set} = \mathtt{s.reliable\_set}[\mathtt{p}]$, it follows that $\mathtt{q} \in \mathtt{s.reliable\_set}[\mathtt{p}]$.

$\mathtt{view}_\mathtt{p}(\mathtt{v})$. The proposition remains true for steps involving $\mathtt{view}_\mathtt{p}(\mathtt{v})$ action because its effect sets $\mathtt{s}'[\mathtt{p}].\mathtt{last\_sent}$ to 0 and because both summands of the right-hand side of the equation also become 0. Indeed, the first summand becomes 0 because CO_RFIFO channels never have messages tagged with views that are larger than the current views of the messages' senders (as can be shown by a simple inductive proof); the second summand becomes 0 because part 2 of Invariant B.4 implies that $\mathtt{s}'[\mathtt{q}].\mathtt{view\_msg}[\mathtt{p}] \neq \mathtt{s}'[\mathtt{p}].\mathtt{current\_view}$.

CO_RFIFO.$\mathtt{deliver}_{\mathtt{p},\mathtt{q}}(\langle \text{`}\mathbf{view\_msg}\text{'}, \mathtt{v}\rangle)$. The proposition remains true for steps involving this action because $\mathtt{s}[\mathtt{q}].\mathtt{view\_msg}[\mathtt{p}] \neq \mathtt{s}[\mathtt{p}].\mathtt{current\_view}$, as follows immediately from Invariant B.4.

CO_RFIFO.$\mathtt{send}_\mathtt{p}(\mathtt{set}, \langle \text{`}\mathbf{app\_msg}\text{'}, \mathtt{m}, \mathtt{Hv}, \mathtt{Hi}\rangle)$ and CO_RFIFO.$\mathtt{deliver}_{\mathtt{p},\mathtt{q}}$ $(\langle \text{`}\mathbf{app\_msg}\text{'}, \mathtt{m}, \mathtt{Hv}, \mathtt{Hi}\rangle)$. For steps involving these actions the truth of the proposition immediately follows from the effects of these actions, the inductive hypotheses, and Invariant B.5. □

The history index attached to an original application message m sent in a view Hv that is in transit on a CO_RFIFO channel to end-point q is equal to the number of such messages (including m) that precede m on that channel plus those (if any) that q has already received.

INVARIANT B.7 (history indices consistency). *In every reachable state* s *of* WV_RFIFO*, for all* Proc p *and* Proc q*, if* $\langle \text{`}\mathbf{app\_msg}\text{'}, \mathtt{m}, \mathtt{Hv}, \mathtt{Hi}\rangle = \mathtt{s.channel}[\mathtt{p}][\mathtt{q}][\mathtt{j}]$ *for some index* j*, then*

$$\mathtt{Hi} = \left|\{\mathtt{msg} \in \mathtt{s.channel}[\mathtt{p}][\mathtt{q}][..\mathtt{j}] \ : \ \mathtt{msg} \in \mathtt{AppMsg\ and\ msg.Hv} = \mathtt{Hv}\}\right| + \begin{cases} \mathtt{s}[\mathtt{q}].\mathtt{last\_rcvd}[\mathtt{p}] & \textit{if } \mathtt{s}[\mathtt{q}].\mathtt{view\_msg}[\mathtt{p}] = \mathtt{Hv}, \\ 0 & \textit{otherwise.} \end{cases}$$

Proof of Invariant B.7. In the initial state $\mathtt{s.channel}[\mathtt{p}][\mathtt{q}]$ is empty. For the inductive step, we consider steps involving the following critical actions.

CO_RFIFO.$\mathtt{lose}(\mathtt{p}, \mathtt{q})$. The proposition remains true since CO_RFIFO.$\mathtt{lose}(\mathtt{p}, \mathtt{q})$ discards only the last messages from the CO_RFIFO $\mathtt{s.channel}[\mathtt{p}][\mathtt{q}]$.

CO_RFIFO.$\mathtt{deliver}_{\mathtt{p},\mathtt{q}}(\langle \text{`}\mathbf{view\_msg}\text{'}, \mathtt{v}\rangle)$. We have to consider the effects on two types of application messages: those associated with view $\mathtt{s}[\mathtt{q}].\mathtt{view\_msg}[\mathtt{p}]$ and those associated with view Hv. Part 1 of Invariant B.4 and Invariant B.5 imply that there are no application messages with $\mathtt{msg.Hv} = \mathtt{s}[\mathtt{q}].\mathtt{view\_msg}[\mathtt{p}]$ on the CO_RFIFO $\mathtt{channel}[\mathtt{p}][\mathtt{q}]$. Thus, the proposition does not apply for such messages. For those messages that have $\mathtt{msg.Hv} = \mathtt{Hv}$, the proposition remains true because $\mathtt{s}'[\mathtt{q}].\mathtt{last\_rcvd}[\mathtt{p}]$ is set to 0 as a result of this action.

CO_RFIFO.$\mathtt{deliver}_{\mathtt{p},\mathtt{q}}(\langle \text{`}\mathbf{app\_msg}\text{'}, \mathtt{m}, \mathtt{Hv}, \mathtt{Hi}\rangle)$. This immediately follows from the effect of this action, the inductive hypothesis, and Invariant B.5.

CO_RFIFO.$\mathtt{send}_\mathtt{p}(\mathtt{set}, \langle \text{`}\mathbf{app\_msg}\text{'}, \mathtt{m}, \mathtt{Hv}, \mathtt{Hi}\rangle)$. The inductive step immediately follows from the inductive hypothesis and Invariant B.6. □

We now prove a generalization of Invariant B.2 which relates application messages either in transit on the CO_RFIFO channels or at end-points' queues to their corresponding messages on the senders' queues.

INVARIANT B.8 (general message consistency). *In every reachable state* s *of* WV_RFIFO*, for all* Procp *and* Procq*, the following are true.*

1. *If* $\langle \text{`}\mathbf{app\_msg}\text{'}, \mathtt{m}, \mathtt{Hv}, \mathtt{Hi}\rangle \in \mathtt{s.channel}[\mathtt{p}][\mathtt{q}]$*, then* $\mathtt{s}[\mathtt{p}].\mathtt{msgs}[\mathtt{p}][\mathtt{Hv}][\mathtt{Hi}] = \mathtt{m}$.
2. *If* $\langle \text{`}\mathbf{fwd\_msg}\text{'}, \mathtt{r}, \mathtt{m}, \mathtt{v}, \mathtt{i}\rangle \in \mathtt{s.channel}[\mathtt{p}][\mathtt{q}]$*, then* $\mathtt{s}[\mathtt{r}].\mathtt{msgs}[\mathtt{r}][\mathtt{v}][\mathtt{i}] = \mathtt{m}$.

3. *If* s[q].msgs[p][v][i] = m, *then* s[p].msgs[p][v][i] = m.

Proof of Invariant B.8. *Basis.* In the initial state all message queues are empty.

Inductive step. The following are the critical actions:

send$_p$(m),
co_rfifo.send$_p$(set, ⟨'**app_msg**', m, Hv, Hi⟩),
co_rfifo.deliver$_{q,p}$(⟨'**app_msg**', m, Hv, Hi⟩),
co_rfifo.send$_p$(set, ⟨'**fwd_msg**', r, v, m, i⟩),
co_rfifo.deliver$_{q,p}$(⟨'**fwd_msg**', r, v, m, i⟩).

For steps involving CO_RFIFO.deliver$_{q,p}$(⟨'**app_msg**', m, Hv, Hi⟩), we use Invariant B.5 and Invariant B.7, which, respectively, imply that history view Hv equals s[p].view_msg[q] and that history index Hi equals s[p].last_rcvd[q] + 1. Inductive steps involving each of the other actions are straightforward. □

Invariant B.2 is a private case of this invariant.

B.2. Virtual synchrony. We now show that automaton VS_RFIFO+TS simulates VSRFIFO : SPEC. We prove this by extending the refinement above using the proof extension theorem of [30] (see Appendix A for details).

B.2.1. Invariants. We prove that end-points that move together from one view to the next consider the same synchronization messages and thus compute the same transitional sets and use the same cuts from the members of the transitional set.

INVARIANT B.9. *In every reachable state* s *of* VS_RFIFO+TS, *for all* Procp, Procq, *and for every* StartIdcid, *if* s[q].sync_msg[p][cid] ≠ ⊥, *then* s[q].sync_msg[p][cid] = s[p].sync_msg[p][cid].

Proof of Invariant B.9. The proposition is true in the initial state s_0 as all s_0[q].sync_msg[p][cid] = ⊥. The inductive step involving a set_cut$_p$() action is trivial, for it only affects the case q = p. The inductive step involving a CO_RFIFO.deliver$_{p,q}$(⟨'**sync_msg**', cid, v, cut⟩) action follows immediately from the following proposition:

⟨'**sync_msg**', cid, v, cut⟩ ∈ s.channel[p][q] ⇒ s[p].sync_msg[p][cid] = ⟨v, cut⟩,

which can be proved by straightforward induction. Indeed, there are two critical actions: CO_RFIFO.send$_p$(set, ⟨'**sync_msg**', cid, v, cut⟩)—immediate from the code—and CO_RFIFO.deliver$_{p,p}$(⟨'**sync_msg**', cid, v, cut⟩)—this may not occur because end-points do not send synchronization messages to themselves. □

COROLLARY B.1. *End-points that move together from one view to the next use the same sets of synchronization messages to calculate transitional sets and message cuts.*

Proof. Consider two end-points that deliver view v′ while in view v. At the time of delivering view v′, each of these end-points has synchronization messages from all end-points in the intersection of these views (second precondition), and these synchronization messages are the same as those at their original end-points (Invariant B.9). Thus, the two end-points calculate the same transitional sets and use the same cuts from the members of this transitional set. □

B.2.2. Simulation. We augment VS_RFIFO+TS with a *global* history variable H_cut that keeps track of the cuts used for moving between views.

For each View v, v′: (Proc → Int)$_\perp$ H_cut[v][v′], initially ⊥

OUTPUT view$_p$(v, T) modifies wv_rfifo.view$_p$(v)
pre: ...

```
eff: ...
     (∀ q ∈ Proc)
       H_cut[current_view][v](q) ← max_{r∈T} (sync_msg[r][v.startId(r)].cut(q)).
```

Variable $\texttt{H\_cut}[\texttt{v}][\texttt{v}']$ is updated every time *any* end-point is delivering view $\texttt{v}'$ while in view $\texttt{v}$. Corollary B.1 implies that whenever this happens after $\texttt{H\_cut}[\texttt{v}][\texttt{v}']$ is set for the first time the value of $\texttt{H\_cut}[\texttt{v}][\texttt{v}']$ remains unchanged.

We now extend the refinement mapping $\texttt{R}()$ of Lemma B.1 with the new mapping $\texttt{R}_\texttt{n}()$:

$$\texttt{For each View v, View v}' : \texttt{R}_\texttt{n}(\texttt{s.H\_cut}[\texttt{v}][\texttt{v}']) = \texttt{cut}[\texttt{v}][\texttt{v}'].$$

We call the resulting mapping $\texttt{R}'()$. We exploit the proof extension theorem of [30] (see Appendix A) in order to prove that $\texttt{R}'()$ is a refinement mapping from VS_RFIFO+TS to VSRFIFO : SPEC.

LEMMA B.2. *Function* $\texttt{R}'()$ *defined above is a refinement mapping from automaton* VS_RFIFO+TS *to automaton* VSRFIFO : SPEC.

Proof of Lemma B.2. *Action correspondence.* The action correspondence is the same as that of WV_RFIFO, except for the steps of the type $(\texttt{s}, \texttt{view}_\texttt{p}(\texttt{v}', \texttt{T}), \texttt{s}')$ which involve VS_RFIFO+TS delivering views to the application clients. Among these steps, those that are the first to set variable $\texttt{H\_cut}[\texttt{v}][\texttt{v}']$ (when $\texttt{s.H\_cut}[\texttt{v}][\texttt{v}'] = \perp$) simulate two steps of VSRFIFO : SPEC: $\texttt{set\_cut}(\texttt{v}, \texttt{v}', \texttt{s}'.\texttt{H\_cut}[\texttt{v}][\texttt{v}'])$ followed by $\texttt{view}_\texttt{p}(\texttt{v}')$. The rest (when $\texttt{s.H\_cut}[\texttt{v}][\texttt{v}'] \neq \perp$) simulate single steps that involve just $\texttt{view}_\texttt{p}(\texttt{v}')$.

Simulation proof. First, we show that the refinement mapping of WV_RFIFO (presented in Lemma B.1) is still preserved after the modifications introduced by VSRFIFO : SPEC to WV_RFIFO : SPEC. Automaton VSRFIFO : SPEC adds the following preconditions to the $\texttt{view}_\texttt{p}(\texttt{v})$ actions of WV_RFIFO : SPEC:

$$\texttt{cut[current\_view[p]][v]} \neq \perp,$$
$$(\forall\ \texttt{q})\quad \texttt{last\_dlvrd[q][p]} = \texttt{cut[current\_view[p]][v](q)}.$$

Since $\texttt{set\_cut}(\texttt{current\_view}[\texttt{p}], \texttt{v}, \texttt{s}'.\texttt{H\_cut}[\texttt{current\_view}[\texttt{p}]][\texttt{v}])$ is simulated before action $\texttt{view}_\texttt{p}(\texttt{v})$, the first precondition holds. The second one follows immediately from the precondition on VS_RFIFO+TS.$\texttt{view}_\texttt{p}(\texttt{v}, \texttt{T})$ and the extended mapping $\texttt{R}'()$.

Second, we show that the mapping $\texttt{R}_\texttt{n}()$ used to extend $\texttt{R}()$ to $\texttt{R}'()$ is also a refinement. For those steps $(\texttt{s}, \texttt{view}_\texttt{p}(\texttt{v}', \texttt{T}), \texttt{s}')$ that are the first to set variable $\texttt{H\_cut}[\texttt{v}][\texttt{v}']$, the action correspondence implies that the mapping is preserved. For those steps that are not the first to set variable $\texttt{H\_cut}[\texttt{v}][\texttt{v}']$, the mapping is preserved because $\texttt{s}'.\texttt{H\_cut}[\texttt{v}][\texttt{v}'] = \texttt{s.H\_cut}[\texttt{v}][\texttt{v}']$, by Corollary B.1. □

From Lemmas B.1 and B.2 and from Theorem A.1, we conclude the following.

THEOREM B.2. VS_RFIFO+TS *implements* VSRFIFO : SPEC *in the sense of trace inclusion.*

B.3. Transitional set. We now show that VS_RFIFO+TS simulates TS : SPEC. The proofs makes use of *prophecy variables.* A simulation proof that uses prophecy variables implies only finite trace inclusion, but this is sufficient for proving safety properties (see Appendix A).

B.3.1. Invariants.

INVARIANT B.10. *In every reachable state* s *of* VS_RFIFO+TS*, for all* Proc p *and for all* StartId id*, if* $\texttt{id} > \texttt{s}[\text{MEMB}].\texttt{start}[\texttt{p}].\texttt{id}$*, then* $\texttt{s}[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{id}] = \perp$.

Proof of Invariant B.10. The proposition is true in the initial state. It remains true for the inductive step involving MEMB.$\texttt{start}_\texttt{p}(\texttt{id}, \texttt{set})$ because $\texttt{s}[\texttt{memb}].\texttt{start}[\texttt{p}].\texttt{id}$ is increased as a result of this action. For the step involving $\texttt{set\_cut}_\texttt{p}()$, the proposition remains true because $\texttt{s}[\texttt{p}].\texttt{start.id} = \texttt{s}[\text{MEMB}].\texttt{start}[\texttt{p}].\texttt{id}$, as implied by the following invariant, which can be proved by straightforward induction.

In every reachable state s of VS_RFIFO+TS, for all Procp, if $\texttt{s}[\texttt{p}].\texttt{start.id} \neq \perp$, then $\texttt{s}[\text{MEMB}].\texttt{start}[\texttt{p}].\texttt{id} = \texttt{s}[\texttt{p}].\texttt{start.id}$. This invariant holds in the initial state. Critical action MEMB.$\texttt{start}_\texttt{p}(\texttt{id}, \texttt{set})$ makes it true; critical action $\texttt{view}_\texttt{p}(\texttt{v}, \texttt{T})$ makes it vacuously true.

Finally, a step involving CO_RFIFO.$\texttt{deliver}_{\texttt{q},\texttt{p}}(\langle \text{`}\textbf{sync\_msg}\text{'}, \texttt{cid}, \texttt{v}, \texttt{cut}\rangle)$ does not affect the proposition because the case $\texttt{q} = \texttt{p}$ cannot happen since end-points do not send synchronization messages to themselves. □

LEMMA B.3. *For any step* $(\texttt{s}, \text{MEMB}.\texttt{start}_\texttt{p}(\texttt{id}, \texttt{set}), \texttt{s}')$ *of* VS_RFIFO+TS,

$$\texttt{s}[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{start.id}] = \perp.$$

Proof of Lemma B.3. The proof follows from the precondition $\texttt{id} > \texttt{s}[\text{MEMB}].\texttt{start}[\texttt{p}].\texttt{id}$ and Invariant B.10. □

INVARIANT B.11. *In every reachable state* s *of* VS_RFIFO+TS, *for all* Procp, *if* $\texttt{s}[\texttt{p}].\texttt{start} \neq \perp$ *and* $\texttt{s}[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{s}[\texttt{p}].\texttt{start.id}] \neq \perp$, *then*

$$\texttt{s}[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{s}[\texttt{p}].\texttt{start.id}].\texttt{view} = \texttt{s}[\texttt{p}].\texttt{current\_view}.$$

Proof of Invariant B.11. The proposition is vacuously true in the initial state. For the inductive step, consider the following critical actions:

MEMB.$\texttt{start}_\texttt{p}(\texttt{id}, \texttt{set})$. The proposition remains vacuously true because $\texttt{s}'[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{start.id}] = \texttt{s}[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{start.id}] = \perp$ (Lemma B.3).

$\texttt{set\_cut}_\texttt{p}()$. This follows immediately from the code.

CO_RFIFO.$\texttt{deliver}_{\texttt{q},\texttt{p}}(\langle \text{`}\textbf{sync\_msg}\text{'}, \texttt{cid}, \texttt{v}, \texttt{cut}\rangle)$. The proposition is unaffected because the case $\texttt{q} = \texttt{p}$ cannot happen since end-points do not send synchronization messages to themselves.

$\texttt{view}_\texttt{p}(\texttt{v})$. The proposition becomes vacuously true because $\texttt{s}'[\texttt{p}].\texttt{start} = \perp$. □

B.3.2. Simulation. We augment automaton VS_RFIFO+TS with a prophecy variable $\texttt{P\_legal\_views}(\texttt{p})(\texttt{id})$ for each Procp and each StartId id. At the time a start id is delivered to an end-point p, this variable is set to a *predicted* finite set of future views that are allowed to contain id as p's start id.

```
Prophecy Variable:
For each Proc p, StartId id: SetOf(View) P_legal_views(p)(id),
                                          initially arbitrary

INTERNAL  MEMB.start_p(id, set)  hidden parameter V, a finite set of views
pre: ...
     choose V such that for all v ∈ V: (p ∈ v.set) ∧ (v.startId(p) = id)
eff: ...
     P_legal_views(p)(id) ← V

OUTPUT  GCS.view_p(v, T)
pre: ...
     (for all q ∈ v.set) v ∈ P_legal_views(q)(v.startId(q))
eff: ...
```

The VS_RFIFO+TS automaton augmented with the prophecy variable has the same traces as those of the original automaton because it is straightforward to show that the following conditions required for adding a prophecy variable hold:

1. Every state has at least one value for $\texttt{P\_legal\_views(p)(id)}$.
2. No step is disabled in the *backward direction* by new preconditions involving P_legal_views.
3. Values assigned to state variables do not depend on the values of P_legal_views.
4. If $\texttt{s}_0$ is an initial state of VS_RFIFO+TS, and $\langle \texttt{s}_0, \texttt{P\_legal\_views} \rangle$ is a state of the automaton VS_RFIFO+TS augmented with the prophecy variable, then this state is an initial state.

INVARIANT B.12. *In every reachable state* s *of* VS_RFIFO+TS*, for all* Proc p*, if* $\texttt{s[p].start} \neq \perp$*, then, for all* View v $\in$ $\texttt{P\_legal\_views(p)(s[p].start.id)}$*, it follows that* $\texttt{p} \in \texttt{v.set}$ *and* $\texttt{v.startId(p)} = \texttt{s[p].start.id}$.

Proof of Invariant B.12. The proof follows by induction. The only critical actions are $\text{MEMB.}\texttt{start}_\texttt{p}(\texttt{id}, \texttt{set})$ and $\texttt{view}_\texttt{p}(\texttt{v}, \texttt{T})$. The proposition is true after the former and is vacuously true after the latter. □

LEMMA B.4. *The following function* TS() *is a* refinement mapping *from automaton* VS_RFIFO+TS *to automaton* TS : SPEC *with respect to their reachable states.*

$$\texttt{TS}(\texttt{s} \in \texttt{ReachableStates}(\text{VS\_RFIFO+TS})) = \texttt{t} \in \texttt{ReachableStates}(\text{TS : SPEC}), \texttt{where}$$

$$\texttt{For eachProcp : t.current\_view[p]} = \texttt{s[p].current\_view}$$

$$\texttt{For eachProcp, Viewv : t.prev\_view[p][v]}$$

$$= \begin{cases} \perp & \textit{if } \texttt{v} \notin \texttt{s.P\_legal\_views[p][v.startId(p)]}, \\ \texttt{s[p].sync\_msg[p][v.startId(p)].view} & \textit{otherwise} \end{cases}$$

Proof of Lemma B.4. *Action correspondence.* A step $(\texttt{s}, \texttt{set\_cut}_\texttt{p}(), \texttt{s}')$ of VS_RFIFO+TS simulates a sequence of steps of TS : SPEC. The sequence consists of steps that involve one $\texttt{set\_prev\_view}_\texttt{p}(\texttt{v}')$ action for each $\texttt{v}' \in \texttt{s.P\_legal\_views(p)}$ $(\texttt{s[p].start.id})$. A step $(\texttt{s}, \texttt{view}_\texttt{p}(\texttt{v}, \texttt{T}), \texttt{s}')$ of VS_RFIFO+TS simulates $(\texttt{TS(s)}, \texttt{view}_\texttt{p}(\texttt{v}, \texttt{T}), \texttt{TS}(\texttt{s}'))$ of TS : SPEC.

Simulation proof. Consider the following critical actions:

$\text{MEMB.}\texttt{start}_\texttt{p}(\texttt{id}, \texttt{set})$. A step involving this action simulates an empty step of TS : SPEC. The simulation holds because $\texttt{s}'\texttt{[p].sync\_msg[p][id]} = \texttt{s[p].sync\_msg[p][id]} = \perp$ (Lemma B.3).

$\texttt{set\_cut}_\texttt{p}()$. This simulates a sequence of steps of TS : SPEC involving one $\texttt{set\_prev\_view}_\texttt{p}(\texttt{v}')$ for each $\texttt{v}' \in \texttt{s.P\_legal\_views(p)(cid)}$, where $\texttt{cid} = \texttt{s[p].start.id}$. Each such step is enabled, as can be seen from the following derivation:

$$\begin{aligned} &\texttt{TS(s).prev\_view[p][v']} \\ &= \texttt{s[p].sync\_msg[p][v'.startId(p)].view} \text{ (Refinement mapping)} \\ &= \texttt{s[p].sync\_msg[p][cid].view} \text{ (Invariant B.12)} \\ &= \perp \text{ (precondition of } \texttt{set\_cut}_\texttt{p}()). \end{aligned}$$

In the poststate, $\texttt{s}'\texttt{[p].sync\_msg[p][cid].view}$ and all $\texttt{TS}(\texttt{s}')\texttt{.prev\_view[p][v']}$ are equal to $\texttt{s[p].current\_view}$; thus the simulation step holds.

$\text{CO\_RFIFO.}\texttt{deliver}_{\texttt{q,p}}(\langle \text{`}\textbf{sync\_msg}\text{'}, \texttt{cid}, \texttt{v}, \texttt{cut} \rangle)$. A step involving this action does not affect any of the variables of the refinement mapping and thus simulates an empty step of TS : SPEC. In particular, note that the case of $\texttt{q} = \texttt{p}$ may not happen because end-points do not send synchronization messages to themselves.

AUTOMATON CLIENT$_p$: SPEC

Signature:

Input: deliver$_p$(q, m), Proc q, AppMsg m
view$_p$(v), View v
block$_p$()

Output: send$_p$(m), AppMsg m
block_ok$_p$()

State: block_status ∈ {unblocked, requested, blocked}, initially unblocked

Transitions:

INPUT **block$_p$**()
eff: block_status ← requested

OUTPUT **block_ok$_p$**()
pre: block_status = requested
eff: block_status ← blocked

OUTPUT **send$_p$**(m)
pre: block_status ≠ blocked
eff: none

INPUT **deliver$_p$**(q, m)
eff: none

INPUT **view$_p$**(v)
eff: block_status ← unblocked

FIG. B.1. *Abstract specification of a blocking client at end-point* p.

view$_p$(v, T). A step involving this action simulates a step of TS : SPEC that involves view$_p$(v, T). The key thing is to show that it is enabled (since it is straightforward to see that, if it is, the refinement is preserved). Action view$_p$(v, T) of TS : SPEC has three preconditions. The fact that they are enabled directly follows from the inductive hypothesis, the code, the refinement mapping, and Invariants B.11 and B.12. □

From Lemma B.4 and Theorem A.1 we conclude the following.

THEOREM B.3. VS_RFIFO+TS *implements* TS : SPEC *in the sense of* finite *trace inclusion.*

B.4. Self-delivery. We now prove that the complete GCS end-point automaton simulates SELF : SPEC. In order to prove this, we need to formalize our assumptions about the behavior of the clients of a GCS end-point: we assume that a client eventually responds to every block request with a block_ok response and subsequently refrains from sending messages until a view is delivered to it. We formalize this requirement by specifying an abstract client automaton in Figure B.1. In this automaton, each locally controlled action is defined to be a task by itself, which means that it eventually happens if it becomes enabled unless it is subsequently disabled by another action.

B.4.1. Invariants. The following invariant states that GCS end-points and their clients have the same perception of what their block_status is.

INVARIANT B.13. *In every reachable state* s *of* GCS, *for all* Proc p, s[GCS$_p$].block_status = s[client$_p$].block_status.

Proof of Invariant B.13. The proof follows by trivial induction. □

INVARIANT B.14. *In every reachable state* s *of* GCS, *for all* Procp, *if* s[p].start ≠ ⊥ *and* s[p].block_status ≠ blocked, *then* s[p].sync_msg[p][s[p].start.id] = ⊥.

Proof of Invariant B.14. In the initial state s_0, s_0[p].start = ⊥; so the proposition is vacuously true. For the inductive step, consider the following critical actions:

MEMB.start$_p$(id, set). The proposition remains true because of Lemma B.3.

block$_p$(). The proposition is true in the poststate if it is true in the prestate.

block_ok$_p$(). The proposition becomes vacuously true because s′[p].block_status = blocked.

set_cut$_p$(). The proposition remains vacuously true because s[p].block_status = s′[p].block_status = blocked.

CO_RFIFO.deliver$_{q,p}$(⟨'**sync_msg**', cid, v, cut⟩). The proposition is unaffected because the case q = p cannot happen since end-points do not send synchronization

messages to themselves.

$\texttt{view}_\texttt{p}(\texttt{v},\texttt{T})$. The proposition becomes vacuously true because $\texttt{s}'[\texttt{p}].\texttt{start} = \bot$. □

INVARIANT B.15. *In every reachable state* s *of* GCS*, for all* Procp*, if* $\texttt{s}[\texttt{p}].\texttt{start} \neq \bot$ *and* $\texttt{s}[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{s}[\texttt{p}].\texttt{start.id}] \neq \bot$*, then* $\texttt{s}[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{s}[\texttt{p}].\texttt{start.id}].\texttt{cut}[\texttt{p}] = \texttt{LastIndexOf}(\texttt{s}[\texttt{p}].\texttt{msgs}[\texttt{p}][\texttt{s}[\texttt{p}].\texttt{current\_view}])$.

Proof of Invariant B.15. In the initial state $\texttt{s}_0$, $\texttt{s}_0[\texttt{p}].\texttt{start} = \bot$, so the proposition is vacuously true. For the inductive step, consider the following critical actions:

$\texttt{send}_\texttt{p}(\texttt{m})$. The proposition is vacuously true because $\texttt{s}'[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{s}[\texttt{p}].\texttt{start.id}] = \bot$, as follows from the precondition $\texttt{s}[\texttt{client}_\texttt{p}].\texttt{block\_status} \neq \texttt{blocked}$ on this action at $\texttt{client}_\texttt{p}$, and from Invariants B.13 and B.14.

$\text{MEMB.}\texttt{start}_\texttt{p}(\texttt{id},\texttt{set})$. The proposition is vacuously true because $\texttt{s}'[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{id}] = \texttt{s}[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{id}]$, which by Lemma B.3 is $\bot$.

$\texttt{set\_cut}_\texttt{p}()$. This follows from $\texttt{p} \in \texttt{current\_view.set}$ (Invariant B.1) and the precondition $(\texttt{forall q} \in \texttt{current\_view.set})\ \texttt{cut}(\texttt{q}) = \texttt{LongestPrefixOf}(\texttt{msgs}[\texttt{q}][\texttt{v}])$.

$\text{CO\_RFIFO.}\texttt{deliver}_{\texttt{q},\texttt{p}}(\langle\text{`}\textbf{sync\_msg}\text{'}, \texttt{cid}, \texttt{v}, \texttt{cut}\rangle)$. The proposition is unaffected because the case $\texttt{q} = \texttt{p}$ cannot happen since, as can be proved by straightforward induction, end-points do not send synchronization messages to themselves.

$\texttt{view}_\texttt{p}(\texttt{v},\texttt{T})$. The proposition becomes vacuously true because $\texttt{s}'[\texttt{p}].\texttt{start} = \bot$. □

B.4.2. Simulation. Lemma B.2 in section B.2 on page 117 establishes function $\texttt{R}'()$ as a refinement mapping from automaton VS_RFIFO+TS to automaton VSRFIFO : SPEC. We now argue that $\texttt{R}'()$ is also a refinement mapping from automaton GCS to automaton SELF : SPEC.

LEMMA B.5. *Refinement mapping* $\texttt{R}'()$ *from automaton* VS_RFIFO+TS *to automaton* VSRFIFO : SPEC *(given in Lemma* B.2*) is also a refinement mapping from automaton* GCS *to automaton* SELF : SPEC*, under the assumption that clients at each end-point* p *satisfy the* $\text{CLIENT}_\texttt{p}$: SPEC *specification for blocking clients.*

Proof. Automaton SELF : SPEC modifies automaton WV_RFIFO : SPEC by adding a precondition, $\texttt{last\_dlvrd}[\texttt{p}][\texttt{p}] = \texttt{LastIndexOf}(\texttt{msgs}[\texttt{p}][\texttt{current\_view}[\texttt{p}]])$, to the steps involving $\texttt{view}_\texttt{p}()$ actions. We have to show that this precondition is enabled when a step of GCS involving $\texttt{view}_\texttt{p}(\texttt{v},\texttt{T})$ attempts to simulate a step of SELF : SPEC involving $\texttt{view}_\texttt{p}(\texttt{v})$. Indeed,

$$
\begin{aligned}
\texttt{s}[\texttt{p}].\texttt{last\_dlvrd}[\texttt{p}] &= \texttt{max}_{\texttt{r}\in\texttt{T}}\texttt{sync\_msg}[\texttt{r}][\texttt{v.startId}(\texttt{r})].\texttt{cut}[\texttt{p}] \text{ (a precondition)}\\
&= \texttt{s}[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{v.startId}(\texttt{p})].\texttt{cut}[\texttt{p}] \text{ (Invariant B.9.)}\\
&= \texttt{s}[\texttt{p}].\texttt{sync\_msg}[\texttt{p}][\texttt{s}[\texttt{p}].\texttt{start.id}].\texttt{cut}[\texttt{p}] \text{ (a precondition)}\\
&= \texttt{LastIndexOf}(\texttt{s}[\texttt{p}].\texttt{msgs}[\texttt{p}][\texttt{s}[\texttt{p}].\texttt{current\_view}]) \text{ (Invariant B.15).}
\end{aligned}
$$

Thus, $\texttt{R}'(\texttt{s}).\texttt{last\_dlvrd}[\texttt{p}][\texttt{p}] = \texttt{LastIndexOf}(\texttt{R}'(\texttt{s}).\texttt{msgs}[\texttt{p}][\texttt{R}'(\texttt{s}).\texttt{current\_view}[\texttt{p}]])$ and the precondition is satisfied. □

From Lemmas B.1, B.2, and B.5 and Theorem A.1 we conclude the following.

THEOREM B.4. *Automaton* GCS *implements automaton* SELF : SPEC *in the sense of trace inclusion, under the assumption that clients at each end-point* p *satisfy the* $\text{CLIENT}_\texttt{p}$: SPEC *specification for blocking clients.*

As a child of VS_RFIFO+TS, GCS also satisfies all the safety properties that VS_RFIFO+TS does, in particular TS : SPEC. Thus, from Theorems B.3 and B.4 we conclude the following.

THEOREM B.5. *Automaton* GCS *implements each of the* WV_RFIFO : SPEC*,*

VSRFIFO : SPEC, TS : SPEC, *and* SELF : SPEC *automata in the sense of trace inclusion, under the assumption that clients at each end-point* p *satisfy the* $\text{CLIENT}_{\mathtt{p}}$: SPEC *specification for blocking clients.*

Appendix C. Correctness proof: Liveness property. In this section we prove that fair executions of our group communication service GCS satisfy liveness property 5.2 of section 5.2. In order to show that a certain action eventually happens, we argue that the preconditions on this action eventually become and stay satisfied, and thus the action eventually occurs, by fairness of the execution. Subsection C.1 below presents a number of invariant that are used in the proof of liveness property 5.2 in subsection C.2.

C.1. Invariants. The following invariant captures the fact that, before an end-point computes who the members of its transitional set are, it does not deliver to its client application messages other than those committed by its own synchronization message. Afterwards, the end-point delivers only the messages committed to delivery by the members of the transitional set.

INVARIANT C.1. *In every reachable state* s *of* GCS*, for all* Proc p*, if* s[p].start $\neq \perp$ *and* s[p].sync_msg[p][s[p].start.id] $\neq \perp$*, then for all* Proc q $\in$ s[p].current_view.set*,*

1. *if* s[p].start.id $\neq$ s[p].memb_view.startId(p)*, then* s[p].last_dlvrd[q] $\leq$ s[p].sync_msg[p][s[p].start.id].cut[q]*;*
2. *otherwise, let* v = s[p].current_view*,* v′ = s[p].memb_view*, and let* T = {q $\in$ v′.set $\cap$ v.set | sync_msg[q][v′.startId(q)].view = v}*; then* s[p].last_dlvrd[q] $\leq max_{\mathtt{r}\in\mathtt{T}}$ s[p].sync_msg[r][v′.startId(r)].cut[q].

Proof of Invariant C.1. The proposition is true in the initial state $\mathtt{s}_0$, since $\mathtt{s}_0$[p].start = $\perp$. For the inductive step, consider the following critical actions:

$\mathtt{deliver}_{\mathtt{p}}(\mathtt{q},\mathtt{m})$. The proposition remains true because the precondition on this action mimics the statement of this proposition.

MEMB.$\mathtt{start}_{\mathtt{p}}(\mathtt{id},\mathtt{set})$. The proposition is vacuously true because s′[p].sync_msg[p][id] = s[p].sync_msg[p][id], which by Lemma B.3 is equal to $\perp$.

MEMB.$\mathtt{view}_{\mathtt{p}}(\mathtt{v})$. In the poststate, s[p].start.id = s[p].memb_view.startId(p), so we must consider the second proposition. Its truth follows from the inductive hypothesis and the fact that p $\in$ T, as implied by Invariant B.1.

$\mathtt{set\_cut}_{\mathtt{p}}()$. The proposition holds since index s[p].last_dlvrd[q] is bounded by LongestPrefixOf(s[p].msgs[q][s[p].current_view]) in every reachable state of the system for any Proc q $\in$ s[p].current_view.set (this fact can be straightforwardly proved by induction) and from the precondition (for all q $\in$ s[p].current_view.set) cut(q) = LongestPrefixOf(s[p].msgs[q][s[p].current_view]).

CO_RFIFO.$\mathtt{deliver}_{\mathtt{q},\mathtt{p}}(\langle$'**sync_msg**', cid, v, cut$\rangle)$. The proposition is unaffected because the case q = p is impossible since end-points do not send cuts to themselves.

$\mathtt{view}_{\mathtt{p}}(\mathtt{v},\mathtt{T})$. The proposition becomes vacuously true because s′[p].start = $\perp$. □

The following invariant states that if an end-point p has end-point q's cut committing certain messages sent by end-point r in view v, then end-point q has those messages buffered.

INVARIANT C.2. *In every reachable state* s *of* GCS*, for all* Proc p*,* Proc q*,* Proc r*, and* StartId cid*, if* s[p].sync_msg[q][cid] $\neq \perp$*, then, for every integer* i *between* 1 *and* s[p].sync_msg[q][cid].cut[r]*,* s[q].msgs[r][s[p].sync_msg[q][cid].view][i] $\neq \perp$.

Proof of Invariant C.2. The truth of the invariant follows from Invariant B.9 if we can prove that an end-point's cut commits the end-point to deliver only those messages that it already has on its msgs queue. Formally, this proposition means

that, in every reachable state s of GCS, for all Proc q, if s[q].start $\neq \perp$ and s[q].sync_msg[q][s[q].start.id] $\neq \perp$, then, for all Proc r and all Int i such that 1 $\leq$ i $\leq$ s[q].sync_msg[q][s[q].start.id].cut[r], s[q].msgs[r][s[q].current_view][i] $\neq \perp$. This proposition can be straightforwardly proved by induction: The only interesting action is $\texttt{set\_cut}_{\texttt{q}}()$. The truth of the proposition after this action is taken follows immediately from the precondition (for all r $\in$ s[q].current_view.set) cut(r) = LongestPrefixOf(s[q].msgs[r][s[q].current_view]). □

INVARIANT C.3. *In every reachable state* s *of* GCS*, for all* Proc p *and* Proc q*, if* q $\in$ s[p].sync_set*, then* (a) q $\in$ s[p].start.set *and* (b) q $\in$ s[p].reliable_set*.*

Proof of Invariant C.3. The proposition is vacuously true in the initial state, where s[p].sync_set is empty. The inductive steps for the critical actions $\text{MEMB.}\texttt{start}_{\texttt{p}}(\texttt{id}, \texttt{set})$, $\text{GCS.}\texttt{view}_{\texttt{p}}(\texttt{v}, \texttt{T})$, and $\text{CO\_RFIFO.}\texttt{send}_{\texttt{p}}(\texttt{set}, \langle \text{`}\textbf{sync\_msg}\text{'}, \texttt{cid}, \texttt{v}, \texttt{cut}\rangle)$ follow immediately from their code in Figure 6.4. The inductive step for the action $\text{CO\_RFIFO.}\texttt{reliable\_set}_{\texttt{p}}(\texttt{set})$ straightforwardly follows from the precondition-effect code in Figures 6.2 and 6.4. The inductive step for the critical action $\text{GCS.}\texttt{set\_cut}_{\texttt{p}}()$ follows from the code, which sets sync_set to {p}, and from the fact that p is always in its own reliable_set and start.set (provided start $\neq \perp$), which can be straightforwardly proved by induction. □

C.2. Liveness proof. The following lemma states that, in any execution of GCS, every $\text{GCS.}\texttt{view}_{\texttt{p}}$ event is preceded by the right $\text{MEMB.}\texttt{view}_{\texttt{p}}$ event, which itself is preceded by the right $\text{MEMB.}\texttt{start}_{\texttt{p}}$ event.

LEMMA C.1. *In every execution sequence* α *of* GCS*, the following are true:*

1. *For every* $\text{GCS.}\texttt{view}_{\texttt{p}}(\texttt{v}, \texttt{T})$ *event, there is a preceding* $\text{MEMB.}\texttt{view}_{\texttt{p}}(\texttt{v})$ *event. Moreover, neither a* $\text{MEMB.}\texttt{start}_{\texttt{p}}$ *nor a* $\text{MEMB.}\texttt{view}_{\texttt{p}}$ *event occurs between* $\text{MEMB.}\texttt{view}_{\texttt{p}}(\texttt{v})$ *and* $\text{GCS.}\texttt{view}_{\texttt{p}}(\texttt{v}, \texttt{T})$.
2. *For every* $\text{MEMB.}\texttt{view}_{\texttt{p}}(\texttt{v})$ *event, there is a preceding* $\text{MEMB.}\texttt{start}_{\texttt{p}}(\texttt{id}, \texttt{set})$ *event with* id = v.startId(p) *and* set $\supseteq$ v.set *such that neither a* $\text{MEMB.}\texttt{start}_{\texttt{p}}$*, nor a* $\text{MEMB.}\texttt{view}_{\texttt{p}}$ *nor a* $\text{GCS.}\texttt{view}_{\texttt{p}}$ *event occurs in* α *between* $\text{MEMB.}\texttt{start}_{\texttt{p}}(\texttt{id}, \texttt{set})$ *and* $\text{MEMB.}\texttt{view}_{\texttt{p}}(\texttt{v})$.

Proof of Lemma C.1.

1. Assume that $\text{GCS.}\texttt{view}_{\texttt{p}}(\texttt{v}, \texttt{T})$ occurs in α. Two of the preconditions on $\text{GCS.}\texttt{view}_{\texttt{p}}(\texttt{v}, \texttt{T})$ are v = p.memb_view and v.startId(p) = p.start.id, which can only become satisfied as a result of a preceding $\text{MEMB.}\texttt{view}_{\texttt{p}}(\texttt{v})$ event, followed by no $\text{MEMB.}\texttt{start}_{\texttt{p}}$ and $\text{MEMB.}\texttt{view}_{\texttt{p}}$ events.
2. Assume that $\text{MEMB.}\texttt{view}_{\texttt{p}}(\texttt{v})$ occurs in α. Then a $\text{MEMB.}\texttt{start}_{\texttt{p}}(\texttt{id}, \texttt{set})$ event with id = v.startId(p) and set $\supseteq$ v.set must precede $\text{MEMB.}\texttt{view}_{\texttt{p}}(\texttt{v})$ because, by the MEMB specification, it is the only possible event that can cause the preconditions for $\text{MEMB.}\texttt{view}_{\texttt{p}}(\texttt{v})$ to become true, and because these preconditions do not hold in the initial state of MEMB.
   There may be several $\text{MEMB.}\texttt{start}_{\texttt{p}}(\texttt{id}, \texttt{set})$ events with the same id and different set arguments. After the last such event, an occurrence of a different $\text{MEMB.}\texttt{start}_{\texttt{p}}$ event or a $\text{MEMB.}\texttt{view}_{\texttt{p}}$ event would violate one of the preconditions of $\text{MEMB.}\texttt{view}_{\texttt{p}}(\texttt{v})$; thus, such events may not happen. As a corollary from this and part 1 of this lemma, a $\text{GCS.}\texttt{view}_{\texttt{p}}(\texttt{v}', \texttt{T}')$ event cannot occur between the last $\text{MEMB.}\texttt{start}_{\texttt{p}}(\texttt{id}, \texttt{set})$ and $\text{MEMB.}\texttt{view}_{\texttt{p}}(\texttt{v})$. □

LEMMA C.2 (liveness). *Let* α *be a fair execution of a group communication service* GCS *in which view* v *becomes eventually stable as defined by Property* 5.1*. Then at each end-point* p $\in$ v.set*,* $\text{GCS.}\texttt{view}_{\texttt{p}}(\texttt{v}, \texttt{T})$*, with some* T*, eventually occurs. Furthermore, for every* $\text{GCS.}\texttt{send}_{\texttt{p}}(\texttt{m})$ *that occurs after* $\text{GCS.}\texttt{view}_{\texttt{p}}(\texttt{v}, \texttt{T})$ *and for every*

$\mathtt{q} \in \mathtt{v.set}$, GCS.$\mathtt{deliver_q(p, m)}$ *also occurs.*

Proof of Lemma C.2. Part I. We first prove that GCS.$\mathtt{view_p(v, T)}$ eventually occurs. Our task is to show that, for each $\mathtt{p} \in \mathtt{v.set}$ and some transitional set T, action GCS.$\mathtt{view_p(v, T)}$ becomes enabled at some point after p receives MEMB.$\mathtt{view_p(v)}$ and that it stays enabled forever thereafter unless it is executed. The fact that α is a fair execution of GCS then implies that GCS.$\mathtt{view_p(v, T)}$ is in fact executed.

In order for GCS.$\mathtt{view_p(v, T)}$ to become enabled, its preconditions (see Figures 6.2 and 6.4) must eventually become and stay satisfied until GCS.$\mathtt{view_p(v, T)}$ is executed. We now consider each of these preconditions:

$\mathtt{v} = \mathtt{p.memb\_view} \neq \mathtt{current\_view}$. This precondition ensures that view v that is attempted to be delivered to the client at p is the latest view produced by MEMB and has not yet been delivered to the client. The precondition becomes satisfied as a result of MEMB.$\mathtt{view_p(v)}$. Since in any reachable state of the system $\mathtt{MEMB.memb\_view} = \mathtt{p.memb\_view} \geq \mathtt{p.current\_view}$ (local monotonicity), this precondition remains satisfied forever, unless GCS.$\mathtt{view_p(v, T)}$ is executed. This is because, by our assumption, α does not contain any subsequent MEMB.$\mathtt{view_p(v')}$, and, hence, by the contrapositive of part 1 of Lemma C.1, it also does not contain any subsequent GCS.$\mathtt{view_p(v', T')}$ with $\mathtt{v'} \neq \mathtt{v}$.

$\mathtt{v.startId(p)} = \mathtt{p.start.id}$. This precondition prevents delivery of obsolete views: it ensures that the MEMB service has not issued a new start notification since the time it produced view v. If this condition is not already satisfied before the last MEMB.$\mathtt{start_p(id, set)}$ event with $\mathtt{id} = \mathtt{v.startId(p)}$ and $\mathtt{set} \supseteq \mathtt{v.set}$, then it becomes satisfied as a result of this event, which, by part 1 of Lemma C.1, must precede MEMB.$\mathtt{view_p(v)}$ in α.

This condition stays satisfied from the time of the last MEMB.$\mathtt{start_p(id, set)}$ at least until GCS.$\mathtt{view_p(v, T)}$ occurs because the only two types of actions, MEMB.$\mathtt{start_p(id', set')}$ and GCS.$\mathtt{view_p(v', T')}$ with $\mathtt{v'} \neq \mathtt{v}$, that may affect the value of p.start cannot occur in α after MEMB.$\mathtt{start_p(id, set)}$, as implied by the assumption on this lemma and Lemma C.1.

$\mathtt{v.set} - \mathtt{sync\_set} = \emptyset$. This precondition ensures that prior to delivering view v, end-point p sends out its synchronization message to every member of v.

Notice that if this precondition becomes satisfied any time after the occurrence of the last MEMB.$\mathtt{start_p(id, set)}$ event with $\mathtt{id} = \mathtt{v.startId(p)}$ and $\mathtt{set} \supseteq \mathtt{v.set}$, then it stays satisfied from then on until GCS.$\mathtt{view_p(v, T)}$ is executed. If the precondition is not already satisfied right after the MEMB.$\mathtt{start_p}$ action, it becomes satisfied as a result of CO_RFIFO.$\mathtt{send_p(set, \langle \text{`}\mathbf{sync\_msg}\text{'}, v.startId(p), v, cut\rangle)}$ with $\mathtt{set} = \mathtt{p.start.set} - \mathtt{p.sync\_set}$. This CO_RFIFO.$\mathtt{send_p}$ action must eventually occur in α because its two preconditions, $(\mathtt{p.sync\_msg[p][id]} \neq \perp)$ and $(\mathtt{set} \subseteq \mathtt{reliable\_set})$, eventually become satisfied, for the following reasons.

1. If the first precondition holds any time after the last MEMB.$\mathtt{start_p(id, set)}$ event with $\mathtt{id} = \mathtt{v.startId(p)}$ and $\mathtt{set} \supseteq \mathtt{v.set}$ occurs, then it stays satisfied from that point on. If it is not already satisfied right after the MEMB.$\mathtt{start_p}$ action, it becomes satisfied as a result of $\mathtt{set\_cut_p()}$. In order for $\mathtt{set\_cut_p()}$ to occur, its precondition, $\mathtt{block\_status} = \mathtt{blocked}$, has to becomes satisfied (see Figure 6.5). This occurs as a result of a $\mathtt{block\_ok_q()}$ input from the client at q. If block_status equals blocked at anytime after MEMB.$\mathtt{start_q(v.startId(q), set)}$, then it remains such until GCS.$\mathtt{view_q(v)}$ happens because $\mathtt{block_q()}$ is not enabled after that, and because GCS.$\mathtt{view_q(v)}$ is the only possible GCS view event (by the contrapositive of part 1 of Lemma C.1). To see that block_status does in fact become blocked, consider the

three possible values of block_status right after MEMB.start_q(v.startId(q), set) occurs:

1. block_status = blocked: We are done.
2. block_status = requested: By Invariant B.13, client.block_ok_q() is enabled. It stays enabled until it is executed because the actions, block_q() and GCS.view_q(), which would disable it, cannot occur. When it is executed, the precondition becomes satisfied.
3. block_status = unblocked: When MEMB.start_q(v.startId(q), set) occurs, block_q() becomes and stays enabled until it is executed. After that, block_status becomes requested and the same reasoning as in the previous case applies.

2. The second precondition, set $\subseteq$ reliable_set, becomes satisfied as a result of action CO_RFIFO.reliable_q(set) with set = current_view.set $\cup$ start.set. This action becomes enabled when q receives MEMB.start_q(v.startId(q), set), and therefore it eventually occurs. Afterwards, reliable_set remains unchanged because CO_RFIFO.reliable_q(set) remains disabled; this is because of the precondition reliable_set $\neq$ set and the fact that q's current_view and start remain unchanged.

When CO_RFIFO.send_p(set, ⟨'**sync_msg**', v.startId(p), v, cut⟩) occurs, p.sync_set is set to p.start.set. Since v.set is a subset of p.start.set, this implies that v.set − p.sync_set eventually becomes and stays $\emptyset$.

(for all q $\in$ v.set $\cap$ p.current_view.set) p.sync_msg[q][v.startId(q)] $\neq \perp$. This precondition ensures that p has received the right synchronization message from every q in v.set $\cap$ p.current_view.set. The argument above implies that q eventually sends to p a synchronization message tagged with v.startId(q) and, at the same time, adds p to q.sync_set, where p remains forever, unless GCS.view_p(v, T) with some T occurs. In order to conclude that CO_RFIFO eventually delivers this synchronization message to p, we argue that, from the time the last synchronization message from q to p is placed on CO_RFIFO.channel[q][p] and at least until it is delivered to p, end-point p is in both CO_RFIFO.reliable_set[q] and CO_RFIFO.live_set[q]. The former implies that CO_RFIFO does not lose any messages (in particular, this synchronization message) from q to p. In conjunction with α being a fair execution, the latter implies that CO_RFIFO eventually delivers every message (in particular, this synchronization message) on the channel from q to p.

1. From the time q sends to p the last synchronization message tagged with v.startId(q) until GCS.view_q(v, T) occurs, p is included in q.sync_set. Invariant C.3 implies that in that period p is included in CO_RFIFO.reliable_set[q]. After GCS.view_q(v, T) occurs, p is still included in CO_RFIFO.reliable_set[q], since p $\in$ v.set.

2. End-point p becomes a member of CO_RFIFO.live_set[q] at the time of MEMB.view_q(v), because MEMB.view_q(v) is linked to CO_RFIFO.live_set_q(v.set) and because p $\in$ v.set. This property remains true afterward because α does not contain any subsequent MEMB events at end-point q.

Thus, end-point p eventually receives the right synchronization messages from every q in v.set $\cap$ p.current_view.set.

last_sent $\geq$ sync_msg[p][v.startId(p)].cut(p). This precondition ensures that before delivering view v, p sends to others all of its own messages indicated in its own cut. This precondition eventually becomes satisfied because sending of application messages via CO_RFIFO.send_p, which increments p.last_sent, is enabled at least until p.last_sent reaches sync_msg[p][v.startId(p)].cut(p), as implied by Invariant C.2.

(for all q∈current_view.set) p.last_dlvrd[q]=$\max_{r\in T}$ p.sync_msg[r][v.startId(r)].cut[q]. This precondition verifies that p has delivered to its client exactly the application messages that it needs to deliver in order for virtually synchronous delivery to be satisfied. By Invariant C.1, the value of p.last_dlvrd[q] never exceeds $\max_{r\in T}$ {p.sync_msg[r][v.startId(r)].cut[q]} for any q. It is therefore left to show that p.last_dlvrd[q] does not remain smaller than $\max_{r\in T}$.

We have shown above that all the other preconditions for delivering view v by p eventually become and remain satisfied until the view is delivered. Consider the part of α after all of these preconditions hold. Let q be an end-point in current_view.set such that p.last_dlvrd[q] < $\max_{r\in T}$p.sync_msg[r][v.startId(r)].cut[q], and let i be p.last_dlvrd[q] + 1. We now argue that p.last_dlvrd[q] eventually becomes i, that is, that p eventually delivers the next message from q. An inductive application of this argument would imply that p.last_dlvrd[q] eventually reaches $\max_{r\in T}$ {p.sync_msg[r][v.startId(r)].cut[q]}.

All the preconditions (except perhaps p.msgs[q][p.current_view][i] $\neq \perp$) for delivering the ith message from q are eventually satisfied because they are the same as the preconditions for p delivering view v, which we have shown to be satisfied. Thus, if the ith message is already on p.msgs[q][p.current_view][i], then delivery of this message eventually occurs by fairness, resulting in p.last_dlvrd[q] being incremented; in this case, we are done.

Therefore, consider the case when p lacks the ith message, m, from q. There are two possibilities:

1. If end-point q is in p's transitional set T for view v, then we know the following:
   1. q's view prior to installing view v is the same as p's current view (by definition of T and Invariant B.11).
   2. q's reliable_set contains p starting before q sent any messages in that view and continuing for the rest of α.
   3. Invariant C.2 implies that q has this message and all the messages that precede it in q.msgs[q][p.current_view].
   4. End-point q is enabled to send these messages to p in FIFO order. The only event that could prevent q from sending these messages is GCS.view_q(v), as it would change the value of q.current_view. However, as we argued above, q must send all of the messages it committed in its cut before delivering view GCS.view_q(v). Self-delivery (Invariant B.15) implies that q's cut includes all of the messages q sent while in v. Thus, q would eventually send m to p.
   5. The fact that the connection between end-points q and p is live at least after MEMB.view_q(v) occurs implies that CO_RFIFO eventually delivers this message to p.
2. Otherwise, if end-point q is not in p's transitional set T for view v, we know by the fact that i is ≤ $\max_{r\in T}$ {p.sync_msg[r][v.startId(r)].cut[q]} that there exist some end-points in T whose synchronization messages commit to deliver the ith message from q in view p.current_view. Let r be an end-point with a smallest identifier among these end-points. Here is what we know:
   1. Invariant C.2 implies that r has this message on its r.msgs[r][p.current_view] queue.
   2. r's reliable_set contains p starting before r sent any messages in that view and continuing for the rest of α.
   3. Upon examination of each of the ForwardingStrategyPredicates in section 6.2.1,

we see that the preconditions for r forwarding the i'th message of q to a set including p eventually become and stay satisfied.

4. Since in both forwarding strategies there is only a finite number of messages from q sent in this view that can be forwarded, fairness implies that the i's message is eventually forwarded to p.
5. The fact that the connection between r and p is live at least after MEMB.$\texttt{view}_\texttt{q}(\texttt{v})$ occurs implies that CO_RFIFO eventually delivers this message to p.

Therefore, the ith message from q is eventually delivered to end-point p, and since, as a result of this, the preconditions on delivering this message to the client at p are satisfied, this delivery eventually occurs, and p.last_dlvrd[q] is incremented. By applying this argument inductively, we conclude that p.last_dlvrd[q] eventually reaches $\texttt{max}_{\texttt{r}\in\texttt{T}}$ p.sync_msg[r][v.startId(r)].cut[q] for every q in current_view.set.

We have shown that each precondition on p delivering GCS.$\texttt{view}_\texttt{p}(\texttt{v},\texttt{T})$ eventually becomes and stays satisfied. Fairness implies that GCS.$\texttt{view}_\texttt{p}(\texttt{v},\texttt{T})$ eventually occurs.

Part II. We now consider the second part of the lemma. The following argument proves that, after GCS.$\texttt{view}_\texttt{p}(\texttt{v},\texttt{T})$ occurs at p, for every subsequent GCS.$\texttt{send}_\texttt{p}(\texttt{m})$ event at p, there is a corresponding GCS.$\texttt{deliver}_\texttt{q}(\texttt{p},\texttt{m})$ event that occurs at every q ∈ v.set:

1. For the rest of α, after GCS.$\texttt{view}_\texttt{p}(\texttt{v},\texttt{T})$ occurs, CO_RFIFO.live_set[p] is equal to v.set. This is true because CO_RFIFO.live_set[p] is set to v.set when MEMB.$\texttt{view}_\texttt{p}(\texttt{v})$ occurs and remains unchanged thereafter because of the assumption that α does not contain any subsequent MEMB events at end-point p.
2. After GCS.$\texttt{view}_\texttt{p}(\texttt{v},\texttt{T})$ occurs and before any CO_RFIFO.$\texttt{send}_\texttt{p}$ event involving a ViewMsg or an AppMsg occurs, p eventually executes CO_RFIFO.$\texttt{reliable}_\texttt{p}(\texttt{v.set})$. Moreover, after that and forever thereafter, both p.reliable_set and CO_RFIFO.reliable_set[p] equal v.set. This is true because GCS.$\texttt{view}_\texttt{p}(\texttt{v},\texttt{T})$ sets p.start to $\perp$ and p.current_view.set to v.set, thus enabling CO_RFIFO.$\texttt{reliable}_\texttt{p}(\texttt{v.set})$. This action eventually happens because α is a fair execution and because for the rest of α there are no subsequent MEMB.$\texttt{start}_\texttt{p}$ and GCS.$\texttt{view}_\texttt{p}(\texttt{v}',\texttt{T}')$ events. Because of the latter reason, p.start and p.current_view.set remain unchanged. Therefore, CO_RFIFO.$\texttt{reliable}_\texttt{p}$ remains disabled and both variables CO_RFIFO.reliable_set[p] and p.reliable_set remain equal to v.set.
   From the above argument and from fairness, it follows that any kind of message that end-point p sends subsequently to q via CO_RFIFO will eventually reach end-point q.
3. Action CO_RFIFO.$\texttt{send}_\texttt{p}(\texttt{v.set} - \{\texttt{p}\}, \langle \text{`}\textbf{view\_msg}\text{'}, \texttt{v}\rangle)$ eventually occurs after action CO_RFIFO.$\texttt{reliable}_\texttt{p}(\texttt{v.set})$ occurs, as follows from the code in Figure 6.4. By the reasoning above, CO_RFIFO delivers this ViewMsg to every end-point q ∈ v.set − {p}, resulting in q.view_msg[p] being set to v for the remainder of α (Invariant B.4).
4. When GCS.$\texttt{send}_\texttt{p}(\texttt{m})$ event occurs at p, m is appended to p.msgs[p][v].
5. After sending the ViewMsg, for the rest of α, if p.msgs[p][v][p.last_sent + 1] contains a message (say m′), action CO_RFIFO.$\texttt{send}_\texttt{p}(\texttt{v.set} - \{\texttt{p}\}, \langle \text{`}\textbf{app\_msg}\text{'}, \texttt{m}'\rangle)$ is enabled, and hence eventually occurs by fairness. Since p.last_sent is incremented after each application message is sent using CO_RFIFO.$\texttt{send}_\texttt{p}$, any message on p.msgs[p][v] is eventually sent to v.set − {p}. As was argued above, these messages are eventually delivered to every end-point q ∈

$\mathtt{v.set} - \{\mathtt{p}\}$. Since $\mathtt{q.view\_msg[p]} = \mathtt{v}$ at the time $\mathtt{q}$ receives $\mathtt{m}'$, $\mathtt{q}$ puts $\mathtt{m}'$ in $\mathtt{q.msgs[p][v][q.last\_rcvd} + 1]$ (Invariant B.5) and increments $\mathtt{q.last\_rcvd}$. Therefore, all messages that end-point $\mathtt{p}$ sends in view $\mathtt{v}$ are eventually inserted with no gaps in the end-point $\mathtt{q}$'s queue, $\mathtt{q.msgs[p][v]}$, for every $\mathtt{q} \in \mathtt{v.set} - \{\mathtt{p}\}$.

6. Once $\text{GCS.}\mathtt{view}_{\mathtt{q}}(\mathtt{v}, \mathtt{T})$ happens (by part I of the proof of the lemma), end-point $\mathtt{q} \in \mathtt{v.set}$ is continuously enabled to deliver a message, $\mathtt{m}'$, from $\mathtt{q.msgs[p][v][q.last\_dlvrd} + 1]$; by fairness, such delivery eventually occurs, resulting in $\mathtt{q.last\_dlvrd[p]}$ being incremented. Therefore, every messages on $\mathtt{q.msgs[p][v]}$ is eventually delivered to the client at $\mathtt{p}$, including the case of $\mathtt{q} = \mathtt{p}$.

It follows from this argument that every $\text{GCS.}\mathtt{send}_{\mathtt{p}}(\mathtt{m})$ event at end-point $\mathtt{p}$ that occurs after $\text{GCS.}\mathtt{view}_{\mathtt{p}}(\mathtt{v}, \mathtt{T})$ in α is eventually followed by a $\text{GCS.}\mathtt{deliver}_{\mathtt{q}}(\mathtt{p}, \mathtt{m})$ at every $\mathtt{q} \in \mathtt{v.set}$. □

Acknowledgments. We thank Alan Fekete, Nancy Lynch, Alex Shvartsman, Jeremy Sussman, and the anonymous referee for helpful suggestions.

REFERENCES

[1] *Comm. ACM*, 39 (4) (1996), special issue on Group Communications Systems.

[2] Y. Amir, D. Breitgand, G. Chockler, and D. Dolev, *Group communication as an infrastructure for distributed system management*, in Proceedings of the 3rd International Workshop on Services in Distributed and Networked Environment (SDNE '96), 1996, IEEE Computer Society, pp. 84–91.

[3] Y. Amir, D. Dolev, S. Kramer, and D. Malki, *Transis: A communication sub-system for high availability*, in Proceedings of the 22nd Symposium on Fault-Tolerant Computing (FTCS '92), 1992, IEEE Computer Society, pp. 76–89.

[4] Y. Amir, D. Dolev, P. M. Melliar-Smith, and L. E. Moser, *Robust and Efficient Replication Using Group Communication*, Technical report CS94-20, Institute of Computer Science, Hebrew University, Jerusalem, Israel, 1994.

[5] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella, *The Totem single-ring ordering and membership protocol*, ACM Trans. Comput. Syst., 13 (1995), pp. 311–342.

[6] T. Anker, G. Chockler, D. Dolev, and I. Keidar, *Scalable group membership services for novel applications*, in Networks in Distributed Computing, M. Mavronicolas, M. Merritt, and N. Shavit, eds., DIMACS Ser. Discrete Math. Theoret. Comput. Sci., AMS, Providence, RI, 1998, pp. 23–42.

[7] T. Anker, G. Chockler, I. Shnaiderman, and D. Dolev, *The Design of Xpand: A Group Communication System for Wide Area Networks*, Technical report 2000-31, Institute of Computer Science, Hebrew University, Jerusalem, Israel, 2000.

[8] T. Anker, D. Dolev, and I. Keidar, *Fault tolerant video-on-demand services*, in Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99), 1999, pp. 244–252.

[9] Ö. Babaoğlu, R. Davoli, and A. Montresor, *Group communication in partitionable systems: Specification and algorithms*, IEEE Trans. Software Engrg., 27 (2001), pp. 308–336.

[10] K. Birman, *Building Secure and Reliable Network Applications*, Manning, Greenwich, CT, 1996.

[11] K. Birman, R. Friedman, M. Hayden, and I. Rhee, *Middleware support for distributed multimedia and collaborative computing*, Software Practice and Experience, 29 (1999), pp. 1285–1312.

[12] K. Birman and T. Joseph, *Exploiting virtual synchrony in distributed systems*, in Proceedings of the 11th ACM SIGOPS Symposium on Operating Systems Principles (SOSP), 1987, pp. 123–138.

[13] K. Birman and R. van Renesse, *Reliable Distributed Computing with the Isis Toolkit*, IEEE Computer Society, Los Alamitos, CA, 1994.

[14] G. Chockler, N. Huleihel, and D. Dolev, *An adaptive totally ordered multicast protocol that tolerates partitions*, in Proceedings of the 17th ACM Symposium on Principles of

Distributed Computing, 1998, pp. 237–246.
[15] G. V. Chockler, *An Adaptive Totally Ordered Multicast Protocol that Tolerates Partitions*, Masters thesis, Institute of Computer Science, Hebrew University, Jerusalem, Israel, 1997.
[16] G. V. Chockler, I. Keidar, and R. Vitenberg, *Group communication specifications: A comprehensive study*, ACM Comput. Surveys, 33 (2001), pp. 1–43.
[17] F. Cristian and F. Schmuck, *Agreeing on Process Group Membership in Asynchronous Distributed Systems*, Technical report CSE95-428, Department of Computer Science and Engineering, University of California, San Diego, CA, 1995.
[18] R. De Prisco, A. Fekete, N. Lynch, and A. Shvartsman, *A dynamic view-oriented group communication service*, in Proceedings of the 17th ACM Symposium on Principles of Distributed Computing, 1998, pp. 227–236.
[19] D. Dolev and D. Malkhi, *The Transis approach to high availability cluster communication*, Comm. ACM, 39 (1996), pp. 64–70.
[20] D. Dolev, D. Malki, and H. R. Strong, *An Asynchronous Membership Protocol that Tolerates Partitions*, Technical report CS94-6, Institute of Computer Science, Hebrew University, Jerusalem, Israel, 1994.
[21] C. Dwork, N. Lynch, and L. Stockmeyer, *Consensus in the presence of partial synchrony*, J. ACM, 35 (1988), pp. 288–323.
[22] A. Fekete, N. Lynch, and A. Shvartsman, *Specifying and using a partitionable group communication service*, ACM Trans. Comput. Syst., 19 (2001), pp. 171–216.
[23] R. Friedman and R. van Renesse, *Strong and Weak Virtual Synchrony in Horus*, Technical report TR 95-1537, Department of Computer Science, Cornell University, Ithaca, NY, 1995.
[24] R. Friedman and A. Vaysburd, *Fast replicated state machines over partitionable networks*, in Proceedings of the 16th IEEE International Symposium on Reliable Distributed Systems (SRDS '97), 1997, pp. 130–137.
[25] R. Guerraoui and A. Schiper, *Consensus: The big misunderstanding*, in Proceedings of the 6th IEEE Computer Society Workshop on Future Trends in Distributed Computing Systems (FTDCS-6), 1997, IEEE Computer Society, 1997, pp. 183–188.
[26] K. Guo, W. Vogels, and R. van Renesse, *Structured virtual synchrony: Exploring the bounds of virtual synchronous group communication*, in Proceedings of the 7th ACM SIGOPS European Workshop, 1996, pp. 213–217.
[27] J. Hickey, N. Lynch, and R. van Renesse, *Specifications and proofs for ensemble layers*, in Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99), Lecture Notes in Comput. Sci. 1579, Springer–Verlag, New York, 1999.
[28] M. Hiltunen and R. Schlichting, *Properties of membership services*, in Proceedings of the 2nd International Symposium on Autonomous Decentralized Systems, 1995, IEEE Computer Society, pp. 200–207.
[29] I. Keidar and D. Dolev, *Efficient message ordering in dynamic networks*, in Proceedings of the 15th ACM Symposium on Principles of Distributed Computing, 1996, pp. 68–76.
[30] I. Keidar, R. Khazan, N. Lynch, and A. Shvartsman, *An inheritance-based technique for building simulation proofs incrementally*, ACM Trans. Software Engrg. and Methodology, 11 (2002), pp. 1–29.
[31] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev, *Moshe: A group membership service for WANs*, ACM Trans. Comput. Syst., 20 (2002), pp. 191–238.
[32] R. Khazan, *A One-Round Algorithm for Virtually Synchronous Group Communication in Wide Area Networks*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 2002.
[33] R. Khazan, A. Fekete, and N. Lynch, *Multicast group communication as a base for a load-balancing replicated data service*, in Proceedings of the 12th International Symposium on Distributed Computing (DISC), Springer-Verlag, New York, 1998, pp. 258–272.
[34] L. Lamport, *Time, clocks, and the ordering of events in a distributed system*, Comm. ACM, 21 (78), pp. 558–565.
[35] B. Lampson, *Generalizing Abstraction Functions*, Principles of Computer Systems class, Handout 8, Laboratory for Computer Science, MIT, Cambridge, MA, 1997; Available via anonymous ftp from ftp://theory.lcs.mit.edu/pub/classes/6.826/www/6.826-top.html.
[36] N. A. Lynch, *Distributed Algorithms*, Morgan–Kaufmann, San Francisco, 1996.
[37] N. A. Lynch and M. Tuttle, *An introduction to input/output automata*, CWI Quarterly, 2 (1989), pp. 219–246.
[38] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal, *Extended Virtual Synchrony*, Technical report ECE93-22, Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA, 1994.

[39] L. Rodrigues, K. Guo, A. Sargento, R. van Renesse, B. Glade, P. Verissimo, and K. Birman, *A dynamic light-weight group service*, in Proceedings of the 15th IEEE International Symposium on Reliable Distributed Systems (SRDS '96), 1996, pp. 23–25; Technical report TR96-1611, Cornell University, Ithaca, New York, 1996.

[40] A. Schiper and A. Ricciardi, *Virtually synchronous communication based on a weak failure suspector*, in Proceedings of the 23rd IEEE Fault-Tolerant Computing Symposium (FTCS '93), 1993, pp. 534–543.

[41] F. B. Schneider, *Implementing fault tolerant services using the state machine approach: A tutorial*, ACM Comput. Surveys, 22 (1990), pp. 299–319.

[42] J. Sussman and K. Marzullo, *The* bancomat *problem: An example of resource allocation in a partitionable asynchronous system*, Theoret. Comput. Sci., to appear.

[43] I. Tarashchanskiy, *Virtual Synchrony Semantics: Client-Server Implementation*, Master of engineering masters thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 2000.

[44] R. van Renesse, K. P. Birman, and S. Maffeis, *Horus: A flexible group communication system*, Comm. ACM, 39 (1996), pp. 76–83.

SIAM J. COMPUT.
Vol. 32, No. 1, pp. 131–151

AN APPROXIMATE L^1-DIFFERENCE ALGORITHM FOR MASSIVE DATA STREAMS*

JOAN FEIGENBAUM†, SAMPATH KANNAN‡, MARTIN J. STRAUSS§, AND MAHESH VISWANATHAN¶

Abstract. Massive data sets are increasingly important in a wide range of applications, including observational sciences, product marketing, and the monitoring and operations of large systems. In network operations, raw data typically arrive in *streams*, and decisions must be made by algorithms that make one pass over each stream, throw much of the raw data away, and produce "synopses" or "sketches" for further processing. Moreover, network-generated massive data sets are often *distributed*: Several different, physically separated network elements may receive or generate data streams that, together, comprise one logical data set; to be of use in operations, the streams must be analyzed locally and their synopses sent to a central operations facility. The enormous scale, distributed nature, and one-pass processing requirement on the data sets of interest must be addressed with new algorithmic techniques.

We present one fundamental new technique here: a space-efficient, one-pass algorithm for approximating the L^1-difference $\sum_i |a_i - b_i|$ between two functions, when the function values a_i and b_i are given as data streams, and their order is chosen by an adversary. Our main technical innovation, which may be of interest outside the realm of massive data stream algorithmics, is a method of constructing families $\{V_j(s)\}$ of limited-independence random variables that are *range-summable*, by which we mean that $\sum_{j=0}^{c-1} V_j(s)$ is computable in time *polylog*(c) for all seeds s. Our L^1-difference algorithm can be viewed as a "sketching" algorithm, in the sense of [Broder et al., *J. Comput. System Sci.,* 60 (2000), pp. 630–659], and our technique performs better than that of Broder et al. when used to approximate the symmetric difference of two sets with small symmetric difference.

Key words. streaming algorithms, distance approximation

AMS subject classifications. 68W20, 68W25

PII. S0097539799361701

1. Introduction.

Massive data sets are increasingly important in a wide range of applications, including observational sciences, product marketing, and the monitoring and operations of large systems. In network operations, raw data typically arrive in *streams*, and decisions must be made by algorithms that make one pass over each stream, throw much of the raw data away, and produce "synopses" or "sketches" for further processing. Moreover, network-generated massive data sets are often *distributed*: Several different, physically separated network elements may receive or generate data streams that, together, comprise one logical data set; to be of use in operations, the streams must be analyzed locally and their synopses sent to a central

*Received by the editors September 14, 1999; accepted for publication (in revised form) July 30, 2002; published electronically December 11, 2002. An extended abstract of this paper appeared in *Proceedings of the* 1999 *IEEE Symposium on Foundations of Computer Science.*

http://www.siam.org/journals/sicomp/32-1/36170.html

†Department of Computer Science, Yale University, New Haven, CT 06520-8285 (feigenbaum@cs.yale.edu). Most of this work was done while this author was a member of the Information Sciences Research Center of AT&T Labs in Florham Park, NJ.

‡Computer and Information Science Department, University of Pennsylvania, Philadelphia, PA 19104-6389 (kannan@cis.upenn.edu). Part of this work was done while this author was visiting AT&T Labs in Florham Park, NJ. This author was supported by grants NSF CCR98-20885 and ARO DAAG55-98-1-0393.

§AT&T Labs–Research, 180 Park Avenue, Florham Park, NJ 07932 (mstrauss@research.att.com).

¶Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 (vmahesh@cs.uiuc.edu). This work was done while this author was a Ph.D. student at the University of Pennsylvania and was supported by grant ONR N00014-97-1-0505, MURI.

operations facility. The enormous scale, distributed nature, and one-pass processing requirement on the data sets of interest must be addressed with new algorithmic techniques.

We present one fundamental new technique here: a space-efficient, one-pass algorithm for approximating the L^1-difference $\sum_i |a_i - b_i|$ between two functions, when the function values a_i and b_i are given as data streams, and their order is chosen by an adversary. This algorithm fits naturally into a toolkit for Internet-traffic monitoring. For example, Cisco routers can now be instrumented with the NetFlow feature [CN98]. As packets travel through the router, the NetFlow software produces summary statistics on each *flow*.[1] Three of the fields in the flow records are source IP-address, destination IP-address, and total number of bytes of data in the flow. At the end of a day (or a week, or an hour, depending on what the appropriate monitoring interval is and how much local storage is available), the router (or, more accurately, a computer that has been "hooked up" to the router for monitoring purposes) can assemble a set of values $(x, f_t(x))$, where x is a source-destination pair, and $f_t(x)$ is the total number of bytes sent from the source to the destination during a time interval t. The L^1-difference between two such functions assembled during different intervals or at different routers is a good indication of the extent to which traffic patterns differ.

Our algorithm allows the routers and a central control and storage facility to compute L^1-differences efficiently under a variety of constraints. First, a router may want the L^1-difference between f_t and f_{t+1}. The router can store a small "sketch" of f_t, throw out all other information about f_t, and still be able to approximate $\|f_t - f_{t+1}\|_1$ from the sketch of f_t and (a sketch of) f_{t+1}.

The functions $f_t^{(i)}$ assembled at each of several remote routers R_i at time t may be sent to a central tape-storage facility C. As the data are written to tape, C may want to compute the L^1-difference between $f_t^{(1)}$ and $f_t^{(2)}$, but this computation presents several challenges. First, each router R_i should transmit its statistical data when R_i's load is low and the R_i-C paths have extra capacity; therefore, the data may arrive at C from the R_i's in an arbitrarily interleaved manner. Also, typically the x's for which $f(x) \neq 0$ constitute a small fraction of all x's; thus, R_i should only transmit $(x, f_t^{(i)}(x))$ when $f_t^{(i)}(x) \neq 0$. The set of transmitted x's is not predictable by C. Finally, because of the huge size of these streams,[2] the central facility will not want to buffer them in the course of writing them to tape (and cannot read from one part of the tape while writing to another), and telling R_i to pause is not always possible. Nevertheless, our algorithm supports approximating the L^1-difference between $f_t^{(1)}$ and $f_t^{(2)}$ at C, because it requires little work space, requires little time to process each incoming item, and can process in one pass all the values of both functions $\{(x, f_t^{(1)}(x))\} \cup \{(x, f_t^{(2)}(x))\}$ in any permutation.

Our L^1-difference algorithm achieves the following performance:

> Consider two data streams of length at most n, each representing the nonzero points on the graph of an integer-valued function on a domain of size n. Assume that the maximum value of either

[1] Roughly speaking, a "flow" is a semantically coherent sequence of packets sent by the source and reassembled and interpreted at the destination. Any precise definition of "flow" would have to depend on the application(s) that the source and destination processes were using to produce and interpret the packets. From the router's point of view, a flow is just a set of packets with the same source and destination IP-addresses whose arrival times at the routers are close enough, for a tunable definition of "close."

[2] In 1999, a WorldNet gateway router generated more that 10Gb of NetFlow data each day.

> function on this domain is M. Then a one-pass streaming algorithm can compute with probability $1-\delta$ an approximation A to the L^1-difference B of the two functions such that $|A-B| \le \epsilon B$, using total space $O(\log(Mn)\log(1/\delta)/\epsilon^2)$ and $O(\log^{O(1)}(Mn)\log(1/\delta)/\epsilon^2)$ time to process each item. The data streams may be interleaved in an arbitrary (adversarial) order. Here space usage is measured in number of bits and time in number of bit operations.

The main technical innovation used in this algorithm is a limited-independence random-variable construction that may prove useful in other contexts:

> A family $\{V_j(s)\}$ of uniform ± 1-valued random variables is called *range-summable* if $\sum_{j=0}^{c-1} V_j(s)$ can be computed in time *polylog*(c) for all seeds s. We construct range-summable families of random variables that are n^2-*bad* 4-*wise independent*.[3]

The property of n^2-bad 4-wise independence suffices for the time- and space-bounds on our algorithm. One can construct a truly 4-wise (in fact, 7-wise) independent range-summable family of random variables based on second-order Reed–Muller codes ([RS99]; for details about second-order Reed–Muller codes, see [MS77]), but the efficiency of the range summation seems to be significantly worse than it is in our construction.

The rest of this paper is organized as follows. In section 2, we give precise statements of our "streaming" model of computation and complexity measures for streaming and sketching algorithms. In section 3, we present our main technical results. Section 4 explains the relationship of our algorithm to other recent work, including that of Broder et al. [BCFM00] on sketching and that of Alon, Matias, and Szegedy [AMS99] and Alon et al. [AGMS99] on frequency moments.

2. Models of computation. Our model is closely related to that of Henzinger, Raghavan, and Rajagopalan [HRR98]. We also describe a related sketch model that has been used, e.g., in [BCFM00].

2.1. The streaming model. As in [HRR98], a *data stream* is a sequence of data items $\sigma_1, \sigma_2, \ldots, \sigma_n$ such that, on each *pass* through the stream, the items are read once in increasing order of their indices. We assume the items σ_i come from a set of size M, so that each σ_i has size $\log M$. In our computational model, we assume that the input stream consists of one or more data streams. We focus on two resources—the *work space* required in bits and the *time to process* each item in the stream. An algorithm will typically also require pre- and postprocessing time, but usually applications can afford more time for these tasks. For the algorithms in this paper, the pre- and postprocessing time is comparable to the per-item time and is not considered further.

DEFINITION 1. *The complexity class* PASST$(s(\delta,\epsilon,n,M), t(\delta,\epsilon,n,M))$ *(to be read as "probably approximately correct streaming space complexity* $O(s(\delta,\epsilon,n,M))$ *and time complexity* $O(t(\delta,\epsilon,n,M))$*") contains those functions* f *on domain* X^n*, where* $|X| = M$*, for which one can output a random variable* R *such that* $|R-f| < \epsilon f$ *with probability at least* $1-\delta$*, and computation of* R *can be done by making a single pass over an instance* $x \in X^n$*, presented in a stream, using total workspace* $O(s(\delta,\epsilon,n,M))$ *and taking time* $O(t(\delta,\epsilon,n,M))$ *to process each item.*

If $s = t$*, we also write* PASST(s) *for* PASST(s,t).

[3] The property of n^2-bad 4-wise independence is defined precisely in section 3 below.

We will also abuse notation and write $A \in \mathrm{PASST}(s,t)$ to indicate that an algorithm A for f witnesses that $f \in \mathrm{PASST}(s,t)$.

Thus f is a function of a single input that has n *elements* or *components*. We allow the input elements of f to be presented in any order in the stream; thus, an input item will be of the form "the jth input element value is a_j." For example, a fragment of the stream representing f might look like $\cdots(5,2)(3,7)(7,4)(2,6)\cdots$, and this is interpreted as $f(5)=2$, $f(3)=7$, etc. Note that the input to f is considered to be static—in a properly formed input stream, at most one item specifies the value of a_j. Thus the length of the input stream is n (items) for our algorithms.[4] Other variants of input streams are possible, in which input values may change (repeatedly) throughout the stream, or in which the input comes in a nonarbitrary order (e.g., in sorted order or random order). We do not consider these variations in this paper.

2.2. The sketch model. Sketches were used in [BCFM00] to check whether two documents are nearly duplicates. A sketch can also be regarded as a *synopsis data structure* [GM99].

Definition 2. *Let X be a set containing at most M items. The complexity class* $\mathrm{PAS}(s(\delta,\epsilon,n,M))$ *(to be read as "probably approximately correct sketch complexity $s(\delta,\epsilon,n,M)$") contains those functions $f : X^n \times X^n \to Z$ of two inputs for which there exists a set S of size $2^{O(s)}$, a randomized* sketch function $h : X^n \to S$, *and a randomized* reconstruction function $\rho : S \times S \to Z$ *such that, for all $x_1, x_2 \in X^n$, with probability at least $1-\delta$, $|\rho(h(x_1),h(x_2)) - f(x_1,x_2)| < \epsilon f(x_1,x_2)$.*

By "randomized function" of k inputs, we mean a function of $k+1$ variables. The first input is distinguished as the source of randomness. It is not necessary that, for all settings of the last k inputs and for most settings of the first input, the function outputs the same value.

Note that we can also define the sketch complexity of a function $f : X \times Y \to Z$ for $X \neq Y$. There may be two different sketch functions involved.

There are connections between the sketch model and the streaming model. Let XY denote the set of concatenations of $x \in X$ with $y \in Y$. It has been noted in [KN97] and elsewhere that a function on XY with low streaming complexity also has low one-round communication complexity (regarded as a function on $X \times Y$), because it suffices to communicate the memory contents of the hypothesized streaming algorithm after reading the X part of the input. Sometimes one can also produce a low-sketch-complexity algorithm from an algorithm with low streaming complexity. Our main result is an example.

Also, in practice, it may be useful for the sketch function h to have low streaming complexity. If the set X is large enough to warrant sketching, then it may also warrant processing by an efficient streaming algorithm.

Formally, we have the following.

Theorem 3. *If $f \in \mathrm{PAS}(s(\delta,\epsilon,n,M))$ via sketch function*

$$h \in \mathrm{PASST}(s(\delta,\epsilon,n,M), t(\delta,\epsilon,n,M)),$$

then $f \in \mathrm{PASST}(2s(\delta,\epsilon,2n,M), t(\delta,\epsilon,2n,M))$, where we identify $f : X^n \times X^n \to Z$ with $f : X^{2n} \to Z$ in the natural way.

[4] It turns out, however, that our algorithms will work if, by convention, we define a_j to be zero if *no* stream item specifies the value of a_j. Thus the length of the input stream may be considerably less than n. Our streaming algorithm for the L^1-distance between two vectors actually, at each point during the stream, can approximate the L^1-distance between the vectors seen thus far, regarding unseen inputs as zero.

We will state our time bounds in terms of field(D), the time necessary to perform a single arithmetic operation in a field of size 2^D. Naïve field-arithmetic algorithms guarantee that field(D) $= O(D^2)$.

3. The L^1-difference of functions.

3.1. Our approach. We consider the following problem. The input stream is a sequence of tuples of the form $(i, a_i, +1)$ or $(i, b_i, -1)$ such that, for each i in the universe $[n]$, there is at most one tuple of the form $(i, a_i, +1)$ and at most one tuple of the form $(i, b_i, -1)$, and a_i and b_i are nonnegative integers. If there is no tuple of the form $(i, a_i, +1)$, then define a_i to be zero for our analysis, and similarly for b_i. Also note that, in general, a small-space streaming algorithm cannot know for which i's the tuple $(i, a_i, +1)$ does not appear. The goal is to approximate the value of $F_1 = \sum |a_i - b_i|$ to within $\pm\epsilon F_1$, with probability at least $1 - \delta$.

Let M be an upper bound on a_i and b_i. We assume that n and M are known in advance; in section 3.7, we discuss small modifications that can be made when either of these is not known in advance.

We first present an intuitive exposition of the algorithm. Suppose that, for each type i, we can define a family of M ± 1-valued random variables $R_{i,j}, j = 0, 1, \ldots, (M-1)$, with independence properties to be specified later. When we encounter a tuple of the form $(i, a_i, +1)$, we add $\sum_{j=0}^{a_i - 1} R_{i,j}$ to a running sum z, and when we encounter a tuple of the form $(i, b_i, -1)$, we subtract $\sum_{j=0}^{b_i - 1} R_{i,j}$ from z. The overall effect on z is to cancel the first $\min(a_i, b_i)$ random variables leaving the sum of the remaining $|a_i - b_i|$ random variables. Finally, consider z^2. There are exactly $\sum_{i=1}^{n} |a_i - b_i|$ terms that are squares of random variables, and these terms contribute exactly the desired quantity F_1 to z^2. If the cross terms $R_{i,j} R_{k,l}$ with $\{i, j\} \neq \{k, l\}$ contribute very little, then z^2 is a good approximation to F_1.

Pairwise independence of the random variables in question will ensure that the expected contribution from these cross terms is 0, and 4-wise independence will ensure that the variance is small, thus ensuring that the cross terms contribute very little with high probability. Therefore, we would ideally like our random variables to be 4-wise independent. In addition, as seen above, we want to be able to compute sums of the form $\sum_{j=0}^{c} R_{i,j}$ efficiently. In order to compute these sums very efficiently, our construction produces random-variable families that deviate slightly from 4-wise independence.

We now develop a more formal treatment of the above. We start with the definition that captures the properties desired of the family of random variables corresponding to one type i. We will show how to construct random variables that satisfy this definition. Later, we extend this to show how to construct random-variable families to handle more than one type.

3.2. Construction of random variable families.

DEFINITION 4. *A family $\{V_j(s)\}$ of uniform ± 1-valued random variables with seed s (chosen at random from some set S of seeds) is called* range-summable, n^2-bad 4-wise independent *if the following properties are satisfied:*

1. *The family $\{V_j(s)\}$ is 3-wise independent, i.e., for all* distinct *j_1, j_2, j_3, for all $a, b, c \in \{+1, -1\}$,*

$$\Pr_s[V_{j_1}(s) = a | V_{j_2}(s) = b \wedge V_{j_3}(s) = c] = \Pr_s[V_{j_1}(s) = a].$$

2. *For all s, $\sum_{j=0}^{c-1} V_j(s)$ can be computed in time polylogarithmic in c.*

3. *For all $a < b$,*

$$E\left[\left(\sum_{j=a}^{b-1} V_j(s)\right)^4\right] = O((b-a)^2).$$

In property 3 *and in similar expressions throughout the rest of this paper, the expectation is computed over s.*

Note that even for 4-wise independent random variables, the sum in property 3 is $\Theta((b-a)^2)$ because of terms of the form $V_j^2(s)V_k^2(s)$. Thus, property 3 does not represent a significant weakening of 4-wise independence. On the other hand, we do not know of a construction using 4-wise independent random variables that matches ours in efficiency with regard to property 2.

We now describe our construction. This is the main technical innovation of our paper. It is also a significant point of departure from the work on frequency moments by Alon et al. [AMS99]. The relationship between our algorithm and the frequency-moment algorithms is explained in section 4.

We will construct a single family of M random variables $V_j(s)$, $0 \le j < M$, such that, for all $c \le M$, one can compute $\sum_{j=0}^{c-1} V_j(s)$ quickly. In the discussion that follows, $\oplus$ represents boolean exclusive-or, and $\vee$ represents boolean or. Logarithms in this paper are always to the base 2.

Suppose, without much loss of generality, that M is a power of 2. Let $H_{(\log M)}$ be the matrix with M columns and $\log M$ rows such that the jth column is the binary expansion of j. For example,

$$H_{(3)} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Let $\hat{H}_{(\log M)}$ be formed from $H_{(\log M)}$ by adding a row of 1's at the top.

$$\hat{H}_{(3)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

We will index the $\log M + 1$ rows of $\hat{H}$ starting with -1 for the row of all 1's, then 0 for the row consisting of the 2^0-bits of the binary expansions, and continue consecutively up to the $(\log(M)-1)$st row. We will left multiply $\hat{H}$ by a seed s of length $\log M + 1$ and use the same indexing scheme for bits of s as for rows of $\hat{H}$. We will also refer to the last bit of s and the last row of $\hat{H}$, where "last" means $(\log M - 1)$st, as the "most significant."

Given a seed $s \in \{0,1\}^{\log M+1}$, let $s \cdot \hat{H}_j$ denote the inner product over $\mathbb{Z}_2$ of s with the jth column of $\hat{H}$. Let i_k denote the coefficient of 2^k in the binary expansion of i. Define $f(i)$ by[5]

$$f(i) = (i_0 \vee i_1) \oplus (i_2 \vee i_3) \oplus \cdots \oplus (i_{\log M-2} \vee i_{\log M-1}). \tag{3.1}$$

[5]Here and henceforth, we will actually assume that M is a power of 4 to simplify the exposition.

Thus, the sequence p of values $f(i)$, $i = 0, 1, 2, \ldots$, is

0111 1000 1000 1000 1000 0111 0111 0111 1000 0111 0111 0111 1000 0111 0111 0111 ...

and can be obtained as the string $p_{\log M}$ by starting with $p_0 = 0$ and putting $p_{k+2} = p_k\,\overline{p_k}\,\overline{p_k}\,\overline{p_k}$, where $\overline{\pi}$ denotes the bitwise negation of the pattern π. Finally, put $V_j(s) = (-1)^{(s\cdot\hat{H}_j)+f(j)}$.

PROPOSITION 5. *The quantity $\sum_{j=0}^{c-1} V_j(s)$ can be computed in time $O(\log(c))$.*

Proof. First assume that c is a power of 4. If $c < M$, then the first c columns of $\hat{H}_{\log M}$ have the form $\binom{\hat{H}_{\log c}}{0}$, and we can reduce our problem to one in which we truncate s to include only the first $1 + \log c$ bits. We may thus assume that $c = M$. Then $\hat{H}_{(\log M)}$ is given recursively by

$$\hat{H}_{(\log M)} = \begin{bmatrix} 1\cdots1 & 1\cdots1 & 1\cdots1 & 1\cdots1 \\ H_{(\log M-2)} & H_{(\log M-2)} & H_{(\log M-2)} & H_{(\log M-2)} \\ 0\cdots0 & 1\cdots1 & 0\cdots0 & 1\cdots1 \\ 0\cdots0 & 0\cdots0 & 1\cdots1 & 1\cdots1 \end{bmatrix}.$$

Also, note that the first M bits of p have the form

$$p_{\log M} = p_{\log M-2}\overline{p_{\log M-2}p_{\log M-2}p_{\log M-2}}.$$

Let s' be a string of length $\log M-2$ that is equal to s without the -1st bit and without the two most significant bits, and let f' denote the fraction of 1's in $s' \cdot H_{(\log M-2)}$. Also, for bits b_1, b_2, let $f_{b_1b_2}$ denote the fraction of 1's in

$$s \cdot \begin{bmatrix} 1\cdots1 \\ H_{(\log M-2)} \\ b_1\cdots b_1 \\ b_2\cdots b_2 \end{bmatrix}.$$

Then $f_{b_1b_2} = f'$ or $f_{b_1b_2} = 1 - f'$, depending on b_1, b_2, and the three bits of s dropped from s' (namely, -1, $\log M - 2$, and $\log M - 1$). Recursively compute f', and use the value to compute all the $f_{b_1b_2}$'s and, from that, the number of 1's in $\sum_{j=0}^{c-1} V_j(s)$. This procedure requires recursive calls of depth that is logarithmic in c.

Similarly, one can compute $\sum_{j=q4^r}^{(q+1)4^r-1} V_j(s)$.

Finally, if c is not a power of 4, write the interval $\{0, \ldots, (c-1)\} = [0, c)$ as the disjoint union of at most $O(\log(c))$ intervals, each of the form $[q4^r, (q+1)4^r)$. Use the above technique to compute the fraction of V's equal to 1 over each subinterval, and then combine. If one is careful to perform the procedure bottom up, the entire procedure requires just $\log(c)$ recursive calls, not $\log^2(c)$ calls. For example, suppose $c = 22$. Write $[0, 22)$ as $[0, 16) \cup [16, 20) \cup [20, 21) \cup [21, 22)$. A naïve way to proceed would be to perform recursive calls 3 deep to compute $\sum_{j=0}^{15} V_j(s)$, then calls 2 deep for $\sum_{j=16}^{19} V_j(s)$, then 1 deep for each of $V_{20}(s)$ and $V_{21}(s)$. It is better to proceed as follows: compute $V_{20}(s)$ directly (by taking the dot product of the first $O(\log(c))$ bits of s with the first $O(\log(c))$ rows of column 20 in $\hat{H}_{\log(M)}$, then adding $f(20)$); use this value to compute $V_{21}(s)$ and $V_{16}(s)$ (each of these computations requires looking at just $O(1)$ bits of s—in this case, $V_{21}(s)$ is the sum of $V_{20}(s)$ and the 2^0 bit of s, and $V_{16}(s)$ is the sum of $V_{20}(s)$ and the 2^2 bit of s); then use $V_{16}(s)$ to compute $\sum_{j=16}^{19} V_j(s)$; and finally use $\sum_{j=16}^{19} V_j(s)$ to compute $\sum_{j=0}^{3} V_j(s)$ and, from that,

$\sum_{j=0}^{15} V_j(s)$. For $j < c$, computing the value of a single $V_j(s)$ takes time $O(\log(c))$, and the overhead in each recursive call takes constant time. Thus, altogether, computing a range sum of V's requires time $O(\log(c))$. □

We now show that this construction yields a family of random variables that is n^2-bad 4-wise independent. The fact that $\{V_j(s)\}$ is three-wise independent is shown in [AS92].

PROPOSITION 6. *For all $a < b$, we have*

$$E\left[\left(\sum_{j=a}^{b-1} V_j(s)\right)^4\right] \leq 5(b-a)^2.$$

Proof. First, note that, for some tuples (j_1, j_2, j_3, j_4), columns j_1, j_2, j_3, and j_4 of $\hat{H}$ are independent. These tuples do not contribute to the expectation on the left of the inequality, because for each desired outcome (v_1, v_2, v_3, v_4), the sets

$$S_{(v_1,v_2,v_3,v_4)} = \{s : (V_{j_1}(s), V_{j_2}(s), V_{j_3}(s), V_{j_4}(s)) = (v_1, v_2, v_3, v_4)\}$$

have the same size by linear algebra.

Second, observe that, because any three columns of $\hat{H}$ are independent, if the columns $\hat{H}_{j_1}, \hat{H}_{j_2}, \hat{H}_{j_3}$, and $\hat{H}_{j_4}$ are dependent, then their mod 2 sum is zero. Thus a dependent tuple has one of 3 basic forms—all four columns are identical; there are two pairs of distinct columns; or all four columns are distinct. In the case of dependent tuples, the seed s is irrelevant to the product $V_{j_1}(s)V_{j_2}(s)V_{j_3}(s)V_{j_4}(s)$ because

$$V_{j_1}(s)V_{j_2}(s)V_{j_3}(s)V_{j_4}(s) \tag{3.2}$$
$$= (-1)^{(s\cdot\hat{H}_{j_1})+f(j_1)} \cdot (-1)^{(s\cdot\hat{H}_{j_2})+f(j_2)} \cdot (-1)^{(s\cdot\hat{H}_{j_3})+f(j_3)} \cdot (-1)^{(s\cdot\hat{H}_{j_4})+f(j_4)}$$
$$= (-1)^{f(j_1)+f(j_2)+f(j_3)+f(j_4)}. \tag{3.3}$$

Line (3.3) follows from the fact that the columns $\hat{H}_{j_1}, \hat{H}_{j_2}, \hat{H}_{j_3}$, and $\hat{H}_{j_4}$ sum to zero. Thus it is sufficient to show that

$$U(a,b) \triangleq \sum_{\substack{a\leq j_1,j_2,j_3,j_4<b \\ j_1\oplus j_2\oplus j_3\oplus j_4=0}} (-1)^{f(j_1)+f(j_2)+f(j_3)+f(j_4)} \leq K(b-a)^2$$

for some constant K. From Theorem 9 below, we can see that $K \leq 5$, and thus the proposition holds. □

We shall now provide upper bounds for the quantity $U(a,b)$ defined in the proof of Proposition 6. We will give two bounds for $U(a,b)$—a simply derived though poor bound (Theorem 8) and a more tediously obtained but much tighter bound (Theorem 9). Before presenting these bounds, we first prove a lemma that is used later in the proofs.

LEMMA 7. $U(4a, 4b) \leq 16U(a,b)$.

Proof. Let (j_1, j_2, j_3, j_4) be a dependent tuple in $[4a, 4b)^4$. Consider the two least significant bits of the j's. We will say that the tuple is *odd* if no two of its members have the same pair of least significant bits; otherwise, we will say that the tuple is *even*. There are 64 possibilities making the columns dependent, because we can choose two bits from each of the first three columns arbitrarily, and this forces a unique choice of the bits from the last column. Of these, $24 = 4!$ are odd and 40 are even. (The 40 even tuples arise from 4 tuples in which all columns have identical bits and 36 tuples

in which the 4 columns are paired in one of 6 ways, and the two pairs are given two distinct values out of the 4 possible in one of 6 ways.)

Note that a dependent tuple is odd if and only if there are an odd number of i's for which $(j_i)_0 \vee (j_i)_1 = 1$. Thus, if (j_1, j_2, j_3, j_4) is an odd dependent tuple and (j_1', j_2', j_3', j_4') is an even dependent tuple where j_i' agrees with j_i on all bits except possibly the two least significant, then the contributions of these two tuples to the $U(4a, 4b)$ cancel out. Therefore, given an odd tuple (j_1, j_2, j_3, j_4), pair it with (j_1', j_2', j_3', j_4') as above. Because $4a$ and $4b$ are multiples of 4, (j_1', j_2', j_3', j_4') will be in the correct range. If (j_1, j_2, j_3, j_4) is a tuple having one of the 16 other (even) configurations of the two least significant bits, attempt to pair it inductively with (j_1', j_2', j_3', j_4') such that j_i and j_i' have the same two least significant bits. Thus $U(4a, 4b) \leq 16U(a, b)$. □

We now give a simple argument that $U(a, b) \leq 27(b-a)^2$.

THEOREM 8. $U(a, b) \leq 27(b-a)^2$.

Proof. Given a and b, find r with $a, b \leq 4^r$. Let α be the smallest multiple of 4 that is at least a and let β be the largest multiple of 4 that is at most b. Then $U(a, b)$ is at most $U(\alpha, \beta)$ plus the number of tuples having at least one column in $[a, \alpha) \cup [\beta, b)$. We will handle the first term inductively; we now count the number of tuples having at least one column in $[a, \alpha) \cup [\beta, b)$. First, there are 4 ways to choose one of j_1, j_2, j_3, and j_4 to be in $[a, \alpha) \cup [\beta, b)$. (Having paid the factor 4, we now call this column j_1.) There are at most 6 ways to choose $j_1 \in [a, \alpha) \cup [\beta, b)$. There are at most $(b-a)^2$ ways to choose j_2 and j_3 in $[a, b)$. Finally, once j_1, j_2, and j_3 are fixed, there is at most one way to choose j_4 to make (j_1, j_2, j_3, j_4) dependent. (Note that j_2, j_3, and j_4 play symmetric roles.) This gives $24(b-a)^2$ tuples altogether. Thus we conclude that

$$\begin{aligned}
U(a,b) &\leq U(4\lceil a/4\rceil, 4\lfloor b/4\rfloor) + 24(b-a)^2 \\
&= 16U(\lceil a/4\rceil, \lfloor b/4\rfloor) + 24(b-a)^2 \\
&\leq 16U(4\lceil\lceil a/4\rceil/4\rceil, 4\lfloor\lfloor b/4\rfloor/4\rfloor) + 24(\lfloor b/4\rfloor - \lceil a/4\rceil)^2 + 24(b-a)^2 \\
&= 16^2 U(\lceil\lceil a/4\rceil/4\rceil, \lfloor\lfloor b/4\rfloor/4\rfloor) + 24(\lfloor b/4\rfloor - \lceil a/4\rceil)^2 + 24(b-a)^2 \\
&\vdots \\
&\leq 16^{\lfloor \log_4(b-a)\rfloor} + 24(b-a)^2\left[1 + 1/16 + 1/16^2 \cdots\right] \\
&\leq (b-a)^2 + 24(b-a)^2(16/15) \\
&\leq 27(b-a)^2,
\end{aligned}$$

as desired. □

We now give a more involved analysis that lets us improve the bound.

THEOREM 9. $U(a, b) \leq 5(b-a)^2$.

Proof. Define $A(a, b)$ to be $\frac{U(a,b)}{(b-a)^2}$. Note that it is sufficient to show that $\sup_{a,b} A(a, b) \leq 5$. We will give a recurrence for $A(a, b)$ and discuss a computer search over a, b with $b-a$ small that yields a bound better than immediately available from the recurrence.

We first assume that $b - a \geq 16$. Let a' be the smallest multiple of 4 that is at least a, and let b' be the greatest multiple of 4 that is at most b. (Because $b - a \geq 16$, it follows that $a \leq a' < b' \leq b$.) The number of unpaired tuples in $[a, b)^4$ is at most the number of unpaired tuples in $[a', b')^4$ plus the number of unpaired tuples having at least one column in $[a, a') \cup [b', b)$. The number of unpaired tuples in $[a', b')^4$ is $U(a', b') \leq 16U(a'/4, b'/4) = A(a'/4, b'/4)(b' - a')^2$. We now count the number of unpaired tuples having at least one column in $[a, a') \cup [b', b)$.

There are at most $36(b-a)$ tuples such that two of the columns are identical and in $[a,a') \cup [b',b)$, and the two other columns are identical. (The four columns may or may not all be equal. Note that a factor $\max\left(\binom{4}{1},\binom{4}{2}\right) = 6$ is needed to assign the four columns to the one or two values.) We now count the tuples whose columns are all different. Pick an assignment of roles for the columns, which contributes a factor 24. There are at most 6 ways to pick j_1 in $[a,a') \cup [b',b)$. Next we will consider the choices of the pair j_2 and j_3, which in turn will determine j_4 uniquely. We will argue that, for most pairs (j_2,j_3), by making local changes to j_2 and j_3, we can produce another tuple of columns (j_1,j_2',j_3',j_4) that cancels out with (j_1,j_2,j_3,j_4). The main difficulty will be to ensure that the columns j_2' and j_3' are in the correct range.

The local change strategy is as follows: Let k be the index of the least significant bit on which j_2 and j_3 disagree. Let k' be the index of the "mate" of k, i.e., the bit that is "or"ed with the kth bit in the computation of $f(j_2)$ or $f(j_3)$. The columns j_2' and j_3' will be obtained by toggling the k'th bit of j_2 and j_3, respectively. We have to check that the tuples (j_1,j_2,j_3,j_4) and (j_1,j_2',j_3',j_4) have opposite parity. To see this, assume without loss of generality that the kth bits of j_2 and j_3 are 0 and 1, respectively. Then the disjunction in expansion (3.1) corresponding to bits k and k' for each of $f(j_3')$ and $f(j_3)$ is 1, because of the 1 in bit k, but the k-k' disjunction for $f(j_2')$ and $f(j_2)$ differ, because the kth bits are zero, but the k'th bits differ. All the other disjunctions are the same in $f(j_2)$ as $f(j_2')$ and in $f(j_3)$ as $f(j_3')$. Note also that (j_1,j_2',j_3',j_4) is a dependent tuple whenever (j_1,j_2,j_3,j_4) is a dependent tuple.

Next we have to determine the conditions for j_2' and j_3' to be between a and b. We will consider the situation in which one of these columns falls below the lower bound a and appeal to symmetry for the situation in which one column is at least b.

For two columns x,y, let $eq(x,y) = \ell$ if the most significant bit in which they differ has index ℓ.

Because we have already paid a factor of 24 for the assignment of roles, choose roles such that $eq(a,j_2) \geq eq(a,j_3)$. Suppose all columns are r bits long. There are at most $2^{r-\ell}$ vectors j_2 for which $eq(a,j_2)$ is ℓ. For each such vector j_2, there are at most $\lceil (b-a)/2^{r-\ell-1} \rceil - 1 \leq (b-a)/2^{r-\ell-1}$ vectors j_3 distinct from j_2 that agree with j_2 on the least significant $r-\ell-1$ bits. (Such (j_2,j_3) pairs may require a toggling of the first ℓ bits that could cause j_2' or j_3' to drop below a.) Thus the total number of problem pairs with respect to the lower bound is at most $2(b-a)$ for each choice of ℓ. Over all choices of ℓ this number is at most $2(b-a)\log(b-a)$. By symmetry, the number of problem pairs with respect to b is also at most $2(b-a)\log(b-a)$. Thus there are at most $4(b-a)\log(b-a)$ ways to pick pairs j_2 and j_3 that do not cancel out. Combining with the 24 ways of assigning roles and the 6 ways of picking j_1 we find that there are at most $576(b-a)\log(b-a)$ tuples that do not get canceled. Thus, including the dependent tuples with repeated columns (at most $36(b-a)$), we get

$$\begin{aligned}
U(a,b) &\leq U(a',b') + 576(b-a)\log(b-a) + 36(b-a) \\
&\leq U(a',b') + 585(b-a)\log(b-a) \\
&\leq 16U(a'/4,b'/4) + 585(b-a)\log(b-a) \\
&= 16A(a'/4,b'/4)(b'/4-a'/4)^2 + 585(b-a)\log(b-a) \\
&= A(a'/4,b'/4)(b'-a')^2 + 585(b-a)\log(b-a) \\
&\leq A(a'/4,b'/4)(b-a)^2 + 585(b-a)\log(b-a),
\end{aligned}$$

and so

$$A(a,b) \leq A(\lceil a/4 \rceil, \lfloor b/4 \rfloor) + 585\frac{\log(b-a)}{(b-a)}.$$

Let

$$D_i = \sup_{4^i < (b-a) \le 4^{i+1}} A(a,b).$$

For each $C \ge 2$, we have the recurrence

$$D_i \le \begin{cases} D_{i-1} + 585 \cdot \frac{2i}{4^i}, & i \ge C, \\ M_C, & i < C, \end{cases} \tag{3.4}$$

where $M_C = \max(D_0, \ldots, D_{C-1})$ is a bound on $A(a,b)$ over $4^i < b - a \le 4^{i+1}$ for $i < C$, i.e., $b - a \le 4^C$. We want to find a minimal solution. We will discuss below how we establish M_C precisely using an exhaustive computer search.

Recurrence (3.4) has a solution

$$\begin{aligned} D_i &= M_C + 585 \sum_{j=C}^{i} \frac{2j}{4^j} \\ &\le M_C + 2 \cdot 585 \frac{C + 1/3}{3 \cdot 4^{C-1}}, \end{aligned}$$

where the empty sum is taken to be zero. If we put $C = 6$, we get

$$D_i \le M_6 + 2.413,$$

whence, for all a, b, $A(a,b) \le M_6 + 2.413$.

It remains to evaluate M_6. We first show that it is sufficient to consider a finite number of pairs $\{a, b\}$ even though the definition of M_6 requires it to be a supremum over an infinite number of pairs. We then use a computer search to find M_6.

CLAIM 10. *The value* $M_C = \max_{b-a \le 4^C = 2^{2C}} A(a,b)$ *is at most*

$$M'_C = \max_{\substack{a \le 2^{2C-1} \\ b \le a + 2^{2C}}} A(a,b).$$

Proof. Suppose (a,b) is a pair with $b - a \le 2^{2C}$ but $a > 2^{2C-1}$. We produce a', b' with $a' < a$ and $b' < b$ such that $b' - a' = b - a$ and $A(a',b') = A(a,b)$. The claim follows.

First, we show that if $a, b \le 2^r$, then $U(a,b) = U(2^r - b, 2^r - a)$. Given a tuple $(j_1, j_2, j_3, j_4) \in [a,b)^4$, write each j with r bits, padding with leading zeros if necessary. Form $j'_i = 2^r - 1 - j_i$ by negating all the bits in j_i. This procedure toggles the parity of the k-k' disjunct in the expansion of $f(j)$ when the k-k' bits are 00 or 11; for each k, in a dependent tuple, there are an even number of columns that are 00 or 11 in bits k and k' and an even number of columns that are 01 or 10 there. It follows that (j_1, j_2, j_3, j_4) and (j'_1, j'_2, j'_3, j'_4) have the same parity. Note also that this mapping is a bijection from $[a,b)$ to $[2^r - b, 2^r - a)$. From this we can conclude that $U(a,b) = U(2^r - b, 2^r - a)$. Similarly, if $a, b \le 3 \cdot 2^r$, then $U(a,b) = U(3 \cdot 2^r - b, 3 \cdot 2^r - a)$.

Finally:

- if $2^{2C-1} < a \le 2^{2C}$, then
  - if $2^{2C-1} < b \le 2^{2C}$, then put $(a', b') = (2^{2C} - b, 2^{2C} - a)$;
  - if $2^{2C} < b \le 3 \cdot 2^{2C-1}$, then put $(a', b') = (3 \cdot 2^{2C-1} - b, 3 \cdot 2^{2C-1} - a)$;

  - if $3 \cdot 2^{2C-1} < b$, then note that $b < a + 2^{2C} \leq 2^{2C+1}$. Put $(a', b') = (2^{2C+1} - b, 2^{2C+1} - a)$.
- if $2^{2C} < a$, then find $q > 2C$ with $2^q < a \leq 2^{q+1}$;
  - if $b \leq 2^{q+1}$, then $2^q < a < b \leq 2^{q+1}$. Put $(a', b') = (2^{q+1} - b, 2^{q+1} - a)$;
  - otherwise, $2^{q+1} < b \leq a + 2^{2C} \leq 2^{q+1} + 2^q = 3 \cdot 2^q$. Put $(a', b') = (3 \cdot 2^q - b, 3 \cdot 2^q - a)$.

In all cases, $a' < a$, $b' < b$, and $b' - a' = b - a$. □

Thus, if we are interested in M_6, i.e., (a, b) with $b - a \leq 4096$, we need only consider $a \leq 2048$. A computer search was done for $A(a, b)$ over $a \leq 2048$ and $b \leq a + 4096$, and the maximum is 2.55334. Thus $A(a, b) \leq 2.56 + 2.413 < 5$. □

3.3. The algorithm. Recall that for the overall algorithm, we will need to generate a family of random variables for each of the different types. It would be ideal to make these families n-wise independent, but that would require storing a seed for each of the n types, which is infeasible. Therefore, we will use short *master seeds* to generate n different seeds that are 4-wise independent and, from these, compute the n families of random variables that we will use to get an estimate of F_1. It will be necessary to repeat this process to achieve the specified values of ϵ and δ.

For each k, $1 \leq k \leq 3\log(1/\delta)$ and for each ℓ, $1 \leq \ell \leq 8A/\epsilon^2$ (where $A = 10$ will be justified later), choose a *master seed* $S_{k,\ell}$ and use $S_{k,\ell}$ to define a 4-wise independent family $\{s_{i,k,\ell}\}$ of n seeds, each of length $\log M + 1$. Each seed $s_{i,k,\ell}$ in turn defines a range-summable, n^2-bad 4-wise independent family $\{V_{i,j,k,\ell}\}$ of M uniform ± 1-valued random variables, where $V_{i,j,k,\ell} \stackrel{\Delta}{=} V_j(s_{i,k,\ell})$.

We can use any standard construction to define a family of seeds from a master seed. For example, we can use the construction based on BCH codes in [AS92]. Another construction is one in which the master seed is used to define the coefficients of a random degree-3 univariate polynomial over a sufficiently large finite field. We will describe and use this more elementary construction.

Let $D = \max(\log M + 1, \log n)$. Choose $F = GF_{2^D}$ as the finite field. Fix a representation for the elements of F as bit strings of length D. Choose a master seed $S_{k,\ell}$ of length $4D$ bits uniformly at random, and view these bits as coefficients a_3, a_2, a_1, a_0 of a degree-3 polynomial $a(x) \in F[x]$. Now define the ith seed, $s_{i,k,\ell} = a(i)$. It is immediate from basic algebra that these seeds are 4-wise independent and that the individual seeds can be computed in a constant number of field operations over the field F.

A final point of concern is whether the use of a master seed to generate individual seeds impacts the analysis of the last subsection. There we assumed that the seed for a single family of random variables was chosen uniformly at random among strings of a fixed length. In our construction here, when the master seed is chosen uniformly at random from strings of the correct length, each seed is also distributed uniformly at random, and hence the analysis of the previous subsection still applies.

A more formal, high-level description of the algorithm is given in Figure 1.

3.4. Correctness. The proof in this section that the algorithm described in Figure 1 is correct closely follows the one given in [AMS99] for the correctness of their algorithm (see section 4.3).

THEOREM 11. *The algorithm described in Figure* 1 *outputs a random variable* $W = \text{median}_k \text{avg}_\ell Z_{k,\ell}^2$ *such that* $|W - F_1| < \epsilon F_1$ *with probability at least* $1 - \delta$.

ALGORITHM L1($\langle (i, c_i, \theta_i) \rangle$).

```
Initialize:
    For k = 1 to 3 log(1/δ) do
        For ℓ = 1 to (8 · A)/ε² do
            //For any A ≥ 10 —see (3.8) and the end of section 3.2
            {  Z_{k,ℓ} = 0
            pick a master seed S_{k,ℓ} from the (k, ℓ)th sample space  }
            // This implicitly defines s_{i,k,ℓ} for 0 ≤ i < n and
            // in turn implicitly defines V_{i,j,k,ℓ} for 0 ≤ i < n and 0 ≤ j < M.

For each tuple (i, c_i, θ_i) in the input stream do
    For k = 1 to 3 log(1/δ) do
        For ℓ = 1 to (8 · A)/ε² do
            Z_{k,ℓ} +=  θ_i Σ_{j=0}^{c_i − 1} V_{i,j,k,ℓ}

Output median_k avg_ℓ Z²_{k,ℓ}.
```

FIG. 1. *High level L^1 algorithm.*

Proof. Note that for each $j < \min(a_i, b_i)$, both $V_{i,j,k,\ell}$ and $-V_{i,j,k,\ell}$ are added to $Z_{k\ell}$, and for $j \geq \max(a_i, b_i)$, neither $V_{i,j,k,\ell}$ nor $-V_{i,j,k,\ell}$ is added. Thus

$$Z_{k\ell} = \sum_i \sum_{\min(a_i,b_i) \leq j < \max(a_i,b_i)} \pm V_{i,j,k,\ell}.$$

We shall now compute $E[Z_{k\ell}^2]$ and $E[Z_{k\ell}^4]$ for each k, ℓ. We shall use the convention that $\sum_{a \leq i < b} = -\sum_{b \leq i < a}$ if $b < a$. For notational convenience, we let $V_{i,j}$ denote $V_{i,j,k,\ell}$ in the analysis below.

$$\begin{aligned}
E[Z_{k,\ell}^2] &= E\left[\left(\sum_i \sum_{j=a_i}^{b_i - 1} V_{i,j}\right)^2\right] \\
&= E\left[\left(\sum_{m=1}^{F_1} \pm V_m\right)^2\right] \qquad (3.5) \\
&= \sum_{m=1}^{F_1} E[(\pm V_m)^2] + 2 \sum_{1 \leq m < m' < F_1} E[(\pm V_m)(\pm V_{m'})] \\
&= \sum_{m=1}^{F_1} 1 \qquad (3.6) \\
&= F_1,
\end{aligned}$$

where, in line (3.5), we have relabeled the indices of V, and, in line (3.6), we have used the pairwise independence of V_m and $V_{m'}$ and the fact that the expectation of each of these random variables is 0.

Next, consider

$$E[Z_{k,\ell}^4] = E\left[\sum_{0\le i_1,i_2,i_3,i_4<n}\ \sum_{\substack{a_{i_1}\le j_1<b_{i_1}\\ a_{i_2}\le j_2<b_{i_2}\\ a_{i_3}\le j_3<b_{i_3}\\ a_{i_4}\le j_4<b_{i_4}}} V_{i_1,j_1}V_{i_2,j_2}V_{i_3,j_3}V_{i_4,j_4}\right].$$

By 3-wise independence and the fact that $E[V^t]=0$ for odd t, the only terms with nonvanishing expectation are of the form $V_{i,j}^4$ (of which there are F_1 terms), $V_{i,j}^2V_{i',j'}^2$ for $(i,j)\neq(i'j')$ (of which there are $\binom{4}{2}F_1(F_1-1)$ terms), and $V_{i_1,j_1}V_{i_2,j_2}V_{i_3,j_3}V_{i_4,j_4}$ for $(i_1,j_1),(i_2,j_2),(i_3,j_3),(i_4,j_4)$ all different. Suppose, in the third case, that i_1,i_2,i_3,i_4 are not all the same. Let $X=\prod_{i_m=i_1}V_{i_m,j_m}$ and $Y=\prod_{i_m\neq i_1}V_{i_m,j_m}$. Then $E[X]=0$ by 3-wise independence of the V's, and X and Y are independent by 4-wise independence of the seeds $s_{i,k,\ell}$. Therefore, if $(i_1,j_1),(i_2,j_2),(i_3,j_3),(i_4,j_4)$ are all different and i_1,i_2,i_3,i_4 are not all the same,

$$E[V_{i_1,j_1}V_{i_2,j_2}V_{i_3,j_3}V_{i_4,j_4}] = E[XY] = 0.$$

Thus we have

$$\begin{aligned} E[Z_{k,\ell}^4] &\le F_1 + 6F_1(F_1-1) + \sum_i E\left[\left(\sum_{j=a_i}^{b_i-1} V_{i,j}\right)^4\right] \\ &\le 6F_1^2 + \sum_i 5(b_i-a_i)^2 \qquad (3.7)\\ &\le 11F_1^2. \end{aligned}$$

In line (3.7), we used Proposition 6, which shows that our construction of random variables is n^2-bad 4-wise independent, with constant 5.

Thus

$$\mathrm{Var}(Z_{k,\ell}^2) = E[Z_{k,\ell}^4] - E^2[Z_{k,\ell}^2] \le A\cdot F_1^2 \qquad (3.8)$$

for $A=10$. Now, put $Y_k = \frac{\epsilon^2}{8\cdot A}\sum_{1\le\ell\le(8\cdot A)/\epsilon^2} Z_{k,\ell}^2$. Then $\mathrm{Var}(Y_k)\le\frac{\epsilon^2}{8}F_1^2$. By Chebyshev's inequality,

$$\begin{aligned} \Pr(|Y_k - F_1| > \epsilon F_1) &\le \frac{\mathrm{Var}(Y_k)}{\epsilon^2 F_1^2} \\ &\le 1/8. \end{aligned}$$

Put $W=\mathrm{median}_k Y_k$. Then $|W-F_1|>\epsilon F_1$ only if we have $|Y_k-F_1|>\epsilon F_1$ for at least half of the k's. Let $A_k=1$ if $|Y_k-F_1|>\epsilon F_1$ and $A_k=0$ otherwise; so, for all k, $E[A_k]\le 1/8$. Put $m=3\log(1/\delta)$ and $A=\sum_{k=1}^m A_k$; then $E[A]\le m/8$. By Chernoff's inequality, the probability that $A\ge m/2\ge(1+3)E[A]$ is at most

$$\begin{aligned} \left[\frac{e^3}{(1+3)^{(1+3)}}\right]^{m/8} &\approx 1.374^{-m} \\ &\le 2^{-m/3} \\ &\le \delta. \end{aligned}$$

The result follows. ∎

3.5. Cost.

THEOREM 12. *There is an implementation of Algorithm* L1 *(in Figure* 1*) that is in*

$$\text{PASST}\left(\log(Mn)\log(1/\delta)/\epsilon^2,\ \text{field}(\log(Mn))\log(1/\delta)/\epsilon^2\right).$$

Proof. The algorithm stores

- $\log(1/\delta)/\epsilon^2$ random variables $Z_{k,\ell}$ (called counters) whose values are at most Mn;
- master seeds, specifying the seeds and, through the seeds, the values of the ± 1-valued random variables $V_{i,j,k,\ell}$.

The space to store the counters is $O(\log(Mn)\log(1/\delta)/\epsilon^2)$. By our construction, for each k, ℓ, we need $O(\max(\log M, \log n)) = O(D)$ bits of master seed, and so we would need $O(D\log(1/\delta)/\epsilon^2)$ bits of storage to store all the master seeds. This is asymptotically the same as the space requirement for storing the counters.

We now consider the cost of processing a single item $(i, c_i, \pm 1)$. First, one has to produce the seeds $s_{i,k,\ell}$ from the master seeds $S_{k,\ell}$. For each i, this involves computing a degree-3 polynomial over $GF(2^D)$, which takes time $O(\text{field}(D))$. From $s_{i,k,\ell}$, we need to compute a range sum $\sum_{j=0}^{b-1} V_j(s_{i,k,\ell})$, which can be computed in $O(\log M)$. Thus, the overall time complexity is $O((\text{field}(D) + \log(M))\log(1/\delta)/\epsilon^2)$. Under the reasonable assumption that field arithmetic takes at least linear time, the first term dominates, and the complexity is as claimed above. ∎

3.6. Optimality. Our algorithm is quite efficient in the parameters n, M, and δ, but it requires space quadratic in $1/\epsilon$. We now show that for some nontrivial settings of M, for all large settings of n, and for all small settings of δ, any sketching algorithm that approximates the L^1-difference to within ϵ requires space close to $1/\epsilon$. Thus, our algorithm uses space within a polynomial of optimal.

THEOREM 13. *Fix $M = 2$. For sufficiently small δ, and for any (large) α and any (small) $\beta > 0$, the L^1-difference problem is not in* $\text{PAS}(\log^\alpha(n)/\epsilon^{1-\beta})$.

A similar result holds in the streaming model.

Proof. We reduce the set-disjointness problem to the L^1-difference problem.

Recall the set-disjointness problem of communication complexity [KN97]. Alice has a string x, $|x| = n$, and Bob has a string y, $|y| = n$, and they want to determine whether there exists a position i such that the ith bit of x and the ith bit of y are both 1. Any protocol for this problem, even a randomized protocol, requires $\Omega(n)$ bits of communication, even under the restriction that there is at most one bit i with $x_i = y_i = 1$. Finally, note that an efficient sketching or streaming algorithm directly yields a protocol with low communication complexity.

Suppose $L^1 \in \text{PAS}(\log^\alpha(n)/\epsilon^{1-\beta})$. Put $\epsilon = 1/(3n)$. Let (a, b) be an instance of set disjointness; so, for all i, $a_i, b_i \le M = 2$. Note that the symmetric difference of a and b as sets is the L^1-difference of a and b as functions. By hypothesis one can, with high probability, approximate the symmetric difference $|a\Delta b|$ to within $\epsilon|a\Delta b| < 1/2$ (whence one can, with high probability, compute $|a\Delta b|$ exactly) using space at most

$$\log^\alpha(n)/\epsilon^{1-\beta} = \log^\alpha(n)(3n)^{1-\beta},$$

i.e., at most $o(n)$. From the exact symmetric difference, one can compute the set intersection size as $(|a| + |b| - |a\Delta b|)/2$ with high probability. This is a contradiction. ∎

3.7. Algorithm for unknown parameters. If M is not known in advance, we won't know in advance how many random variables to construct (or, equivalently, how many bits are needed for each seed). Note that, because of the recursive construction of H and p, this is not a problem. If at any point we encounter a tuple (i_0, c, θ) with c bigger than the maximum C encountered before, we simply do the following. For each k, ℓ, we pick $(\log c - \log C)$ new random bits, which we associate with the master seed $S_{k,\ell}$. Use the new random bits to extend randomly (when needed) each of the n seeds $s_{i,k,\ell}$ to length $\lceil \log c \rceil + 1$. We also virtually form larger matrices $\hat{H}$ and pattern p without actually instantiating them.

It is also possible to run the algorithm with a constant factor space penalty if n is not known in advance. Initialize n to 2. Pick a field size appropriate for this n and for the known (or so far encountered) value of M. Let F_1 be this field and f_1 be its size. Start reading the stream, performing calculation in F_1. At an arbitrary point in the stream, suppose we are using a field F_c of size f_c.

If we now read a tuple (i, c_i, θ) where i is too big to handle in F_c, we compute the smallest q such that a degree q extension of F_c is sufficient to handle i, and for each k, ℓ, we prepare a new master seed in this extension which we denote by F_{c+1}. Note that $f_{c+1} = f_c^q$. In other words, for each k, ℓ, we generate at random the coefficients of a degree-3 polynomial in F_{c+1}. The new master seeds and field are to be used for all types which could not be handled in F_c but can be handled in F_{c+1}.

By keeping track of all field sizes we use and all master seeds for each of these field sizes, when we encounter a new type we can easily figure out the field size needed to handle this type using the correct seeds. The union of all seeds is still 4-wise independent. At the end, we will have a final value of n (the maximum type), and, along the way, in the worst case, we will have constructed master seeds for families of size $\nu, \nu^{1/2}, \nu^{1/4}, \nu^{1/8}, \ldots$, where $\nu^{1/2} < n \leq \nu$. For each k, ℓ, storing these master seeds and successive fields requires storage space $\log(\nu), \frac{1}{2}\log(\nu), \frac{1}{4}\log(\nu) \ldots$, and, thus, the storage for the master seeds is $O(\max(\log M, \log \nu)) = O(\max(\log M, \log n))$. The total space for all the master seeds and fields is $O((\max(\log M, \log n)) \log(1/\delta)/\epsilon^2)$. The space required for storing the counters remains $O(\log(Mn) \log(1/\delta)/\epsilon^2)$, and so the overall space is $O(\log(Mn) \log(1/\delta)/\epsilon^2)$. The average processing time per item increases by $o(1) \cdot \text{field}(\log(Mn))$, but preparing the last new collection of master seeds takes time $\log(\nu) \log(1/\delta)/\epsilon^2$, and this time represents an (acceptable) additive increase in the maximum per-item time. Note that the amortized per-item time is asymptotically the same as for the case when n is known in advance.

4. Related work.

4.1. Relationship with sketch algorithms. In [BCFM00], the authors consider the problem of detecting near-duplicate web pages. For their purpose and ours, a web page is a subset of a large universe, and two web pages A and B are near-duplicates if $r(A, B) = \frac{|A \cap B|}{|A \cup B|}$ is large. They present an algorithm that computes a small fixed-size "sketch" of each web page such that, with high probability, $r(A, B)$ can be approximated to within additive error given the two sketches. A central technique is based on the observation that, under a random injection h of the universe into the integers, the probability that the minimal element of $h(A \cup B)$ is in $h(A \cap B)$ is exactly $r(A, B)$. (In practice, the injections come from a small sample space; for the purpose of our comparison, we can consider truly random injections.) Some of the relevant techniques in [BCFM00] were used earlier in [C97, BGMZ97].

Our results on computing the L^1-difference between two functions can be viewed

TABLE 1
Relative-error approximability via sketches.

	$\|A\|$, $\|B\|$	$\|A \cap B\|$	$\|A \cup B\|$	$\|A \Delta B\|$
Here	trivial	iff large	yes	yes
[BCFM00]	trivial	iff large	yes	iff large

as a sketch algorithm. The sketch function h takes as input the graph of a single function and performs the algorithm of section 3, getting a set $\{Z_{k,\ell}\}$ of random variables. (Note that the same master seeds must be used for all sketches.) To reconstruct the L^1-difference from two sketches $\{Z_{k,\ell}\}, \{Z'_{k,\ell}\}$, compute $\rho(\{Z_{k,\ell}\}, \{Z'_{k,\ell}\}) = \text{median}_k \text{avg}_\ell (Z_{k,\ell} - Z'_{k,\ell})^2$.

THEOREM 14. *The L^1-difference of two functions from $\{0, \ldots, n-1\}$ to $\{0, \ldots, M-1\}$ is in*

$$\text{PAS}(\log(Mn) \log(1/\delta)/\epsilon^2).$$

In particular, the L^1-difference (or L^2-difference) of two characteristic functions χ_A and χ_B is the size of the symmetric difference $|A \Delta B|$; we've shown how to approximate it to within small relative error with high probability. The size of the sketch is $O(\log(Mn) \log(1/\delta)/\epsilon^2)$, the space bound of the streaming algorithm. Finally, note that computation of these sketches can be performed in the streaming model, which is sometimes an advantage both theoretically and in practice.

COROLLARY 15. *The symmetric difference between two sets from a universe of size n is in*

$$\text{PAS}(\log(n) \log(1/\delta)/\epsilon^2).$$

One can now ask which cells of the A-B Venn diagram can be approximated as functions of (A, B) in the sketch model using our techniques and using the techniques of [BCFM00]. First note that $|A|$, $|B|$, and $|A| + |B|$ are trivial in the sketch model. Next, an additive approximation of $r = r(A, B)$ yields an approximation of $(1+r)$ and, thus, of $1/(1+r)$ with small relative error; thus, $|A \cup B| = (|A| + |B|)/(1+r)$ can be approximated with small relative error using the techniques of [BCFM00] or with small relative error as $(|A| + |B| + |A \Delta B|)/2$ using our techniques. In general, one cannot approximate $|A \cap B|$ with small relative error, even using randomness [KN97], but, if $|A \cap B|$ is sufficiently large compared with $|A \cup B|$, the intersection can be approximated as $|A \cap B| = (|A| + |B|)r/(1+r)$ by [BCFM00] and as $|A \cap B| = (|A| + |B| - |A \Delta B|)/2$ by our methods. Finally, the techniques of [BCFM00] only approximate $1 - r$ additively, and, if $1 - r$ is smaller than the error ϵ of approximation (i.e., if $|A \Delta B| < \epsilon |A \cup B|$), then the techniques of [BCFM00], which approximate $|A \Delta B|$ as $|A \Delta B| = (|A| + |B|)\frac{1-r}{1+r}$, do not perform well,[6] but our technique approximates $|A \Delta B|$ with small relative error regardless of the size of $|A \Delta B|$.

This information is summarized in Table 1. Other cells in the Venn diagram reduce to these results, e.g., by complementing A or B. (Note that A and A-complement may have different sizes.)

[6] The results of [BCFM00] are the best possible in their original context, which is somewhat different from our context.

4.2. Approximating sizes of supports and the zeroth frequency moment. In this section, we briefly consider three variants.

Let

$$\begin{aligned} F_0^{\neq} &= |\{i : a_i \neq b_i\}|, \\ F_0^{\neq 0} &= |\{i : (a_i = 0 \wedge b_i \neq 0) \vee (a_i \neq 0 \wedge b_i = 0)\}|, \\ F_0^{20\%} &= |\{i : (a_i > 1.2b_i) \vee (b_i > 1.2a_i)\}|. \end{aligned}$$

Note that these are all generalizations of $F_0 = |\{i : a_i \neq 0\}|$, which was studied in [AMS99]. We will show that $F_0^{\neq}$ and $F_0^{\neq 0}$ can be approximated, but $F_0^{20\%}$ cannot be approximated. We do this by using reductions under which

$$\mathrm{PASST}(\log(M)\log(n)\log(1/\delta)/\epsilon^2)$$

is closed.

To approximate $F_0^{\neq}$, put

$$A_{i,x} = \begin{cases} 1 & a_i = x, \\ 0 & \text{otherwise} \end{cases}$$

and

$$B_{i,x} = \begin{cases} 1 & b_i = x, \\ 0 & \text{otherwise.} \end{cases}$$

Then $\frac{1}{2}\sum_{i,x} |A_{i,x} - B_{i,x}| = F_0^{\neq}$, where the sum is over $0 \leq i < n$ and $0 \leq x < M$. We can approximate this L^1-difference.

To approximate $F_0^{\neq 0}$, put

$$A_i = \begin{cases} 1 & a_i > 0, \\ 0 & \text{otherwise} \end{cases}$$

and

$$B_i = \begin{cases} 1 & b_i > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Then $\sum_i |A_i - B_i| = F_0^{\neq 0}$. This is used in section 4.1.

Finally, consider $F_0^{20\%}$. We reduce the set-disjointness problem (restricted to inputs with intersection size at most one) to that of approximating $F_0^{20\%}$.

Let (x, y) be an instance of set disjointness, and put

$$a_i = \begin{cases} 11 & x_i = 1, \\ 13 & \text{otherwise} \end{cases}$$

and

$$b_i = \begin{cases} 15 & y_i = 1, \\ 13 & \text{otherwise.} \end{cases}$$

Then a_i and b_i differ by at least 20% exactly in the 11-15 case, i.e., exactly when $x_i = y_i = 1$.

If we could output a number X such that $|X - F_0^{20\%}| \leq \epsilon F_0^{20\%}$ for $\epsilon < 1$ with probability $1 - \delta$, then we would be able to distinguish the situation $F_0^{20\%} = 0$ from $F_0^{20\%} = 1$ and in turn distinguish $|x_i \cap y_i| = 0$ from $|x_i \cap y_i| = 1$, a contradiction.

Note that we cannot even output a random variable X satisfying the following apparently (but not actually) weaker condition:

$$F_0^{21\%}(1-\epsilon) \leq X \leq F_0^{20\%}(1+\epsilon),$$

with $F_0^{21\%} = |\{i : (a_i > 1.21b_i) \vee (b_i > 1.21a_i)\}|$, because, in fact,

$$\{i : (a_i > 1.21b_i) \vee (b_i > 1.21a_i)\} = \{i : (a_i > 1.20b_i) \vee (b_i > 1.20a_i)\},$$

so that $F_0^{21\%} = F_0^{20\%}$.

In summary, putting $F_0^\tau = |\{i : (a_i > (1+\tau)b_i) \vee (b_i > (1+\tau)a_i)\}|$, we obtain the following.

THEOREM 16. *We have*

1. $F_0^{\neq} \in \mathrm{PASST}(\log(Mn)\log(1/\delta)/\epsilon^2,\ \mathrm{field}(\log(Mn))\log(1/\delta)/\epsilon^2)$;
2. $F_0^{\neq 0} \in \mathrm{PASST}(\log(Mn)\log(1/\delta)/\epsilon^2,\ \mathrm{field}(\log(Mn))\log(1/\delta)/\epsilon^2)$;
3. *for all* $\tau \in (0,1)$, *all fixed* $\epsilon < 1$, $\delta < 1/4$, *and* $M > 1/\tau + 2$, *and, for any* $f = o(n)$, $F_0^\tau \notin \mathrm{PASST}(f(n))$.

Also,

1. $F_0^{\neq} \in \mathrm{PAS}(\log(Mn)\log(1/\delta)/\epsilon^2,\ \mathrm{field}(\log(Mn))\log(1/\delta)/\epsilon^2)$;
2. $F_0^{\neq 0} \in \mathrm{PAS}(\log(Mn)\log(1/\delta)/\epsilon^2,\ \mathrm{field}(\log(Mn))\log(1/\delta)/\epsilon^2)$;
3. *for all* $\tau \in (0,1)$, *all fixed* $\epsilon < 1$, $\delta < 1/4$, *and* $M > 1/\tau + 2$, *and, for any* $f = o(n)$, $F_0^\tau \notin \mathrm{PAS}(f(n))$.

4.3. Approximating the L^2-difference and the second frequency moment. In [AMS99], the authors consider the following problem. The input is a sequence of elements from $[n] = \{0, \ldots, n-1\}$. An element $i \in [n]$ may occur many times. We let a_i denote the number of times i occurs. As above, assume that, for all i, $|a_i| \leq M$.

The kth frequency moment F_k of the sequence is defined to be $\sum a_i^k$. Note that the first frequency moment $F_1 = \sum a_i$ is just the length of the stream and is therefore trivial to compute, but other frequency moments are nontrivial. Alon, Matias, and Szegedy [AMS99] give a variety of upper and lower bounds for frequency moments. In particular, for $F_2 = \sum a_i^2$, they show the following.

THEOREM 17 (see [AMS99]).

$$F_2 \in \mathrm{PASST}((\log(Mn))\log(1/\delta)/\epsilon^2,\ \mathrm{field}(\log(Mn))\log(1/\delta)/\epsilon^2).$$

We sketch the algorithm, without proof, in order to illustrate the previous work that is our point of departure. A full treatment of the correctness and work space of the algorithm of Theorem 17 may be found in [AMS99].

Proof (sketch). For each k, $1 \leq k \leq \Theta(\log(1/\delta))$, and for each ℓ, $1 \leq \ell \leq \Theta(1/\epsilon^2)$, let $\{v_{k\ell}[i]\}_i$ be a set of 4-wise independent ± 1-valued random variables. Output

$$\mathrm{median}_k \mathrm{avg}_\ell \left(\sum a_i v_{k\ell}[i]\right)^2. \qquad \square$$

Consider now a generalization of the input allowing signed examples, which were also considered by Alon et al. in [AGMS99]. That is, each item in the sequence consists of a type $i \in [n]$ and a sign $\pm$, and there may be many items of type i of each

sign. Denote by a_i the number of positive occurrences of i and by b_i the number of negative occurrences of i, and let L_k denote $\sum |a_i - b_i|^k$.

The following corollary was obtained independently by Alon et al. [AGMS99].

COROLLARY 18. $L_2 \in \text{PASST}(\log(Mn)\log(1/\delta)/\epsilon^2,\ \text{field}(\log(Mn))\log(1/\delta)/\epsilon^2)$.

Proof (sketch). With k, ℓ, and $v_{k\ell}[i]$ as above, output

$$\text{median}_k \text{avg}_\ell \left(\sum (a_i - b_i) v_{k\ell}[i]\right)^2. \qquad \square$$

The algorithm of Corollary 18 can be used to approximate the L^2-difference between the two functions a and b.

Note that for signed input examples, computing the frequency moment L_1 is nontrivial, modulo the special case of when $a_i \geq b_i$ for all i.

For any p, the problem of computing the pth frequency moment and that of computing the corresponding L^p-difference of functions differs only in the representation of the input stream. Given an L^p-instance stream $\langle (i, c_i, \theta_i) \rangle$, one can expand each item (i, c_i, θ_i) into c_i occurrences of (θ, i) to get a frequency moment instance. Therefore a frequency moment algorithm for signed examples can be used to compute the L^p-difference of functions, but note that, in general, one pays a high cost in processing time, even just to read the input—the input has been expanded exponentially. The algorithm of Corollary 18 avoids this cost, because it is efficient in both input representations. Thus, the L^2-difference is in

$$\text{PASST}((\log(Mn))\log(1/\delta)/\epsilon^2,\ \text{field}(\log(Mn))\log(1/\delta)/\epsilon^2).$$

4.4. Earlier work on probabilistic counting. In [FM83], the authors give a small-space randomized algorithm that approximates the number of distinct elements in a stream. Their algorithm assumed the existence of certain ideal hash functions. Later, [AMS99] improved this result by substituting a practically available family of hash functions. [AMS99] also gives a variety of other results on approximating the frequency moments. Many results of this kind, some old and some new, are described in [GM99].

Acknowledgments. We thank S. Muthukrishnan for helpful discussions. We also thank an anonymous referee for a very careful reading of the paper.

REFERENCES

[AMS99] N. ALON, Y. MATIAS, AND M. SZEGEDY, *The space complexity of approximating the frequency moments,* J. Comput. System Sci., 58 (1999), pp. 137–147.

[AGMS99] N. ALON, P. GIBBONS, Y. MATIAS, AND M. SZEGEDY, *Tracking join and self-join sizes in limited storage*, in Proceedings of the 18th Symposium on Principles of Database Systems, ACM, New York, 1999, pp. 10–20.

[AS92] N. ALON AND J. SPENCER, *The Probabilistic Method*, John Wiley, New York, 1992.

[BCFM00] A. BRODER, M. CHARIKAR, A. FRIEZE, AND M. MITZENMACHER, *Min-wise independent permutations,* J. Comput. System Sci., 60 (2000) pp. 630–659.

[BGMZ97] A. BRODER, S. GLASSMAN, M. MANASSE, AND G. ZWEIG, *Syntactic clustering of the Web*, in Proceedings of the Sixth International World Wide Web Conference, Santa Clara, CA, World Wide Web Consortium, Cambridge, MA, 1997, pp. 391–404. Also available online from http://www.scope.gmd.de/info/www6/technical/paper205/paper205.html.

[CN98] CISCO NETFLOW, 1998. Available online from http://www.cisco.com/warp/public/732/netflow/.

[C97] E. COHEN, *Size-estimation framework with applications to transitive closure and reachability,* J. Comput. System Sci., 55 (1997), pp. 441–453.

[FM83] P. Flajolet and G. N. Martin, *Probabilistic counting*, in Proceedings of the 24th Foundations of Computer Science Conference, IEEE Computer Society, Los Alamitos, 1983, pp. 76–82.

[GM99] P. Gibbons and Y. Matias, *Synopsis Data Structures for Massive Data Sets*, in Eternal Memory Algorithms, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 50, AMS, Providence, RI, 1999, pp. 39–70.

[HRR98] M. R. Henzinger, P. Raghavan, and S. Rajagopalan, *Computing on Data Streams,* Technical Report 1998-011, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, 1998.

[KN97] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.

[MS77] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Math. Library 16, North Holland, New York, 1977.

[RS99] E. Rains and N. J. A. Sloane, *Private communication.*

SIAM J. COMPUT.
Vol. 32, No. 1, pp. 152–171

LIMIT LAWS FOR SUMS OF FUNCTIONS OF SUBTREES OF RANDOM BINARY SEARCH TREES*

LUC DEVROYE†

Abstract. We consider sums of functions of subtrees of a random binary search tree and obtain general laws of large numbers and central limit theorems. These sums correspond to random recurrences of the quicksort type, $X_n \stackrel{\mathcal{L}}{=} X_{I_n} + X'_{n-1-I_n} + Y_n$, $n \geq 1$, where I_n is uniformly distributed on $\{0, 1, \ldots, n-1\}$, Y_n is a given random variable, $X_k \stackrel{\mathcal{L}}{=} X'_k$ for all k, and, given I_n, X_{I_n} and X'_{n-1-I_n} are independent. Conditions are derived such that $(X_n - \mu n)/\sigma\sqrt{n} \stackrel{\mathcal{L}}{\to} \mathcal{N}(0,1)$, the normal distribution, for some finite constants μ and σ.

Key words. binary search tree, data structures, probabilistic analysis, limit law, convergence, toll functions, Stein's method, random trees

AMS subject classifications. 68Q25, 68P05, 60F05

PII. S0097539701383923

1. Introduction.

In this note, we consider a random binary search tree with n nodes obtained by inserting, in the standard manner, the values $\sigma_1, \ldots, \sigma_n$ of a random permutation of $\{1, \ldots, n\}$ into an initially empty tree. Equivalently, the search tree is obtained by inserting n independently and identically distributed (i.i.d.) uniform $[0,1]$ random variables $X_1, \ldots, X_n$. Most shape-related quantities of the tree have been well studied, including the expected depth and the exact distribution of the depth of X_n (Knuth (1973), Lynch (1965)), the limit theory for the depth (Mahmoud and Pittel (1984), Devroye (1988)), the first two moments of the internal path length (Sedgewick (1983)), the limit theory for the height of the tree (Pittel (1984), Devroye (1986), (1987)), and various connections with the theory of random permutations (Sedgewick (1983)) and the theory of records (Devroye (1988)). Surveys of known results can be found in Vitter and Flajolet (1990), Mahmoud (1992), and Gonnet (1984). Search trees are also useful in the analysis of quicksort. For recurrences of the quicksort type, we have $X_n \stackrel{\mathcal{L}}{=} X_{I_n} + X'_{n-1-I_n} + f(n)$, $n \geq 1$, $f(n) > 0$ for some $n > 0$, $f(0) = 0$, where I_n is uniformly distributed on $\{0, 1, \ldots, n-1\}$. X_n represents the number of comparisons in quicksort, and $f(n) = n - 1$. Other choices for $f(.)$ are of importance elsewhere. It is not hard to see that X_n is identical to the sum over all nodes u in a random binary search tree of $f(N(u))$, where $N(u)$ is the size of the subtree at u. The purpose of this note is to obtain central limit theorems for this class of random variables, regardless of the choice of f within a large class of functions.

In general, one might study the following class of tree parameters for random binary search trees: Let f be a mapping from the space of all permutations to the real line, and set

$$X_n = \sum_u f(S(u)),$$

*Received by the editors January 22, 2001; accepted for publication (in revised form) April 3, 2002; published electronically December 11, 2002. This reseach was was sponsored by NSERC grant A3456 and by FCAR grant 90-ER-0291.

http://www.siam.org/journals/sicomp/32-1/38392.html

†School of Computer Science, McGill University, 3480 University Street, Montreal, Canada H3A 2K6 (luc@cs.mcgill.ca).

where $S(u)$ is the random permutation associated with the subtree rooted at node u in the random binary search tree. More precisely, model a random binary search tree as follows. We let $U_1, \ldots, U_n$ be i.i.d. uniform $[0,1]$-valued random variables, and construct the unique binary search tree for $(1, U_1), \ldots, (n, U_n)$ with the property that

(i) it is a random binary search tree with respect to the first coordinates in the pairs, and

(ii) it is a heap with respect to the second coordinates, which can be regarded as time stamps, with increasing values as one travels from the root down any path.

A permutation is clearly described by any subset of $(1, U_1), \ldots, (n, U_n)$. It is this unique description we follow. For example, the root of the binary search tree contains that pair (i, U_i) with the smallest U_i value, the left subtree contains all pairs (j, U_j) with $j < i$, and the right subtree contains those pairs with $j > i$. Each node u can thus (recursively) be associated with a subset $S(u)$ of $(1, U_1), \ldots, (n, U_n)$. The pair that sticks with u is that with the smallest second component in $S(u)$.

With this embedding and representation, X_n is a sum over all nodes of a certain function of the permutation associated with each node. This definition is very broad. As each permutation uniquely determines subtree shape, a special case includes the functions of subtree shapes.

EXAMPLE 1 (the toll functions). *In the first class of applications, we let $N(u)$ be the size of the subtree rooted at u (thus, if u is the overall root, $N(u) = n$) and set $f(S(u)) = g(|S(u)|)$. Define*

$$X_n = \sum_u g(N(u)) \ .$$

Examples of such tree parameters abound:

A. *If $g(n) \equiv 1$ for $n > 0$, then $X_n = n$.*

B. *If $g(n) = \mathbb{1}_{[n=k]}$ for fixed $k > 0$, then X_n counts the number of subtrees of size k.*

C. *If $g(n) = \mathbb{1}_{[n=1]}$, then X_n counts the number of leaves.*

D. *If $g(n) = n - 1$ for $n > 1$, then X_n counts the number of comparisons in classical quicksort. Note, however, that $g(n)$ grows too rapidly for us to be able to apply the theorem below.*

E. *If $g(n) = \log_2 n$ for $n > 0$, then X_n is the logarithm base two of the product of all subtree sizes.*

F. *If $g(n) = \mathbb{1}_{[n=1]} - \mathbb{1}_{[n=2]}$ for $n > 0$, then X_n counts the number of nodes in the tree that have two children, one of which is a leaf.*

EXAMPLE 2 (tree patterns). *Fix a tree T. We write $S(u) \approx T$ if the subtree at u defined by the permutation $S(u)$ is equal to T, where equality of trees refers to shape only, not node labeling. Note that at least one, and possibly many permutations with $|S(u)| = |T|$, may give rise to T. If we set*

$$X_n = \sum_u \mathbb{1}_{[S(u) \approx T]},$$

then X_n counts the number of subtrees precisely equal to T. Note that these subtrees are necessarily disjoint. We are tempted to call them suffix tree patterns, as they hug the bottom of the binary search tree.

EXAMPLE 3 (prefix tree patterns). *Fix a tree T. We write $S(u) \supset T$ if the subtree at u defined by the permutation $S(u)$ consists of T (rooted now at u) and possibly other*

nodes obtained by replacing all external nodes of T by new subtrees. Define

$$X_n = \sum_u \mathbb{1}_{[S(u) \supset T]}.$$

For example, if T is a single node, then X_n counts the number of nodes, n. If T is a complete subtree of size $2^{k+1} - 1$ and height k, then X_n counts the number of occurrences of this complete subtree pattern (as if we try to count by sliding the complete tree to all nodes in turn to find a match). Matching complete subtrees can possibly overlap. If T consists of a single node and a right child, then X_n counts the number of nodes in the tree with just one right child.

EXAMPLE 4 (imbalance parameters). *If we set $f(S(u))$ equal to 1 if and only if the sizes of the left and right subtrees of u are equal, then X_n counts the number of nodes at which we achieve a complete balance.*

EXAMPLE 5 (local counters). *Following notation introduced by Devroye* (1991), *we may just elect to study indicator functions f with $f(S(u)) = 0$ if $|S(u))| > k$ for a fixed given k. In fact, the setting in Devroye* (1991) *is more general, as permutations are not necessarily restricted to those that correspond to nodes in the binary search tree.*

In this paper, we study X_n. First we derive its mean and variance. This is followed by a weak law of large numbers for X_n/n. Several interesting examples illustrate this universal law. A general central limit theorem with normal limit is obtained for X_n using Stein's method. Several specific laws are obtained for particular choices of f. For example, for toll functions g as in Example 1, with $g(n)$ growing at a rate inferior to $n^{1/3}$, a universal central limit theorem is established in Theorem 6.

2. Another representation of binary search trees. We replace the sum over all nodes u in a random tree in the definition of X_n by a sum over a deterministic set of index pairs, thereby greatly facilitating systematic analysis. We denote by $\sigma(i,k)$ to subset $(i,U_i),\ldots,(i+k-1,U_{i+k-1})$, so that $|\sigma(i,k)| = k$. We define $\sigma^*(i,k) = \sigma(i-1,k+1)$, with the convention that $(0,U_0) = (0,0)$ and $(n+1,U_{n+1}) = (n+1,0)$. Define the event

$$A_{i,k} = [\sigma(i,k) \text{ defines a subtree}] \ .$$

This event depends only on $\sigma^*(i,k)$, as $A_{i,k}$ happens if and only if among $U_{i-1},\ldots,U_{i+k}$, U_{i+k}, U_{i-1} and U_{i+k} are the two smallest values. We set $Y_{i,k} = \mathbb{1}_{[A_{i,k}]}$ and note that it is a function of $U_{i-1},\ldots,U_{i+k}$. Rewrite our tree parameter as follows:

$$X_n = \sum_u f(S(u)) = \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} Y_{i,k} f(\sigma(i,k)) \ .$$

For example, in the toll function example with toll function g, this yields

$$X_n = \sum_u g(|S(u)|) = \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} Y_{i,k} g(k) \ .$$

3. Mean and variance for toll functions. Let σ be a uniform random permutation of size k. Then define

$$\mu_k = \mathbb{E}\{f(\sigma)\} \ ,$$

$$\tau_k^2 = \mathbb{E}\{f^2(\sigma)\} \ ,$$

and

$$M_k = \sup_{\sigma:|\sigma|=k} |f(\sigma)| \ .$$

Note that $|\mu_k| \le \tau_k \le M_k$. In the toll function example, we have $\mu_k = g(k)$ and $\tau_k = M_k = |g(k)|$. We opt to develop the theory below in terms of these parameters. For some parts, such as the law of large numbers, the second moment approach may be avoided, but this comes at the expense of considerably more intricate computations and proofs.

LEMMA 1. *Assume $|\mu_k| < \infty$ for all k, $\mu_k = o(k)$, and*

$$\sum_{k=1}^{\infty} \frac{|\mu_k|}{k^2} < \infty \ .$$

Define

$$\mu = \sum_{k=1}^{\infty} \frac{2\mu_k}{(k+2)(k+1)} \ .$$

Then

$$\lim_{n\to\infty} \frac{\mathbb{E}\{X_n\}}{n} = \mu \ .$$

If also $|\mu_k| = O(\sqrt{k}/\log k)$, then $\mathbb{E}\{X_n\} - \mu n = o(\sqrt{n})$.

Proof. We have

$$\begin{aligned}\mathbb{E}\{X_n\} &= \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \mathbb{E}\{Y_{i,k}\}\mu_k \\ &= \sum_{i=2}^{n} \sum_{k=1}^{n-i} \frac{2}{(k+2)(k+1)}\mu_k + \sum_{k=1}^{n-1} \frac{1}{k+1}\mu_k + \sum_{i=1}^{n} \frac{1}{n-i+2}\mu_{n-i+1} + \mu_n,\end{aligned}$$

since

$$\mathbb{E}\{Y_{i,k}\} = \begin{cases} 1 & \text{if } i = 1 \text{ and } i+k = n+1; \\ 1/(k+1) & \text{if } i = 1 \text{ or } i+k = n+1 \text{ but not both}; \\ 2/(k+2)(k+1) & \text{otherwise.} \end{cases}$$

It is trivial to conclude the first part of Lemma 1. For the last part, we have

$$\begin{aligned}&|\mathbb{E}\{X_n - \mu n\}| \\ &\le \sum_{k=1}^{\infty} \frac{2}{(k+2)(k+1)}|\mu_k| + \sum_{i=2}^{n} \sum_{k=n-i+1}^{\infty} \frac{2}{(k+2)(k+1)}|\mu_k| + 2\sum_{k=1}^{n} \frac{|\mu_k|}{k+1} + |\mu_n| \\ &\le O(1) + \sum_{k=1}^{\infty} \frac{2\min(k,n)|\mu_k|}{(k+2)(k+1)} + 2\sum_{k=1}^{n} \frac{|\mu_k|}{k+1} + |\mu_n| \\ &\le O(1) + 4\sum_{k=1}^{n} \frac{|\mu_k|}{k+1} + n\sum_{k=n+1}^{\infty} \frac{|\mu_k|}{(k+2)(k+1)} + |\mu_n| \ . \qquad \square\end{aligned}$$

LEMMA 2. *Assume that $M_n < \infty$ for all n and that $f \geq 0$. Assume that for some $b \geq c \geq a > 0$, we have $\mu_n = O(n^a)$, $\tau_n = O(n^c)$, and $M_n = O(n^b)$. If $a + b < 2$, $c < 1$, then $\mathbb{V}\{X_n\} = o(n^2)$. If $a + b < 1$, $c < 1/2$, then $\mathbb{V}\{X_n\} = O(n)$. If f is a toll function and $M_n = O(n^b)$, then $\mathbb{V}\{X_n\} = o(n^2)$ if $b < 1$ and $\mathbb{V}\{X_n\} = O(n)$ if $b < 1/2$.*

Proof. Let Z_α, $\alpha \in A$, be a finite collection of random variables with finite second moments. Let E denote the collection of all pairs (α, β) from A^2 with $\alpha \neq \beta$ and Z_α not independent of Z_β. If $S = \sum_{\alpha \in A} Z_\alpha$, then

$$\mathbb{V}\{S\} = \sum_{\alpha \in A} \mathbb{V}\{Z_\alpha\} + \sum_{(\alpha,\beta) \in E} \left(\mathbb{E}\{Z_\alpha Z_\beta\} - \mathbb{E}\{Z_\alpha\}\mathbb{E}\{Z_\beta\}\right).$$

We apply this fact with A being the collection of all pairs (i, k), with $1 \leq i \leq n$ and $1 \leq k \leq n-i+1$. Let our collection of random variables be the products $Y_{i,k} f(\sigma(i, k))$, $(i, k) \in V$. Note that E consists only of pairs $((i, k), (j, \ell))$ from A^2 with $i + k \geq j - 1$ and $j + \ell \geq i$. This means that the intervals $[i, i + k - 1]$ and $[j, j + \ell - 1]$ correspond to an element of E if and only if they overlap or are disjoint and separated by exactly zero or one integer m. But to bound $\mathbb{V}\{X_n\}$ from above, since $f \geq 0$, we have

$$\mathbb{V}\{X_n\} \leq \sum_{(i,k) \in A} \mathbb{V}\{Y_{i,k} f(\sigma(i, k))\} + \sum_{((i,k),(j,\ell)) \in E} \mathbb{E}\{Y_{i,k} f(\sigma(i, k)) Y_{j,\ell} f(\sigma(j, \ell))\} = \text{I} + \text{II}.$$

By the independence of $Y_{i,k}$ and $f(\sigma(i, k))$, we have

$$\begin{aligned}
\mathbb{V}\{Y_{i,k} f(\sigma(i, k))\} &= \mathbb{V}\{Y_{i,k}\}\mathbb{E}\{f^2(\sigma(i, k))\} + (\mathbb{E}\{Y_{i,k}\})^2 \mathbb{V}\{f(\sigma(i, k))\} \\
&= \mathbb{V}\{Y_{i,k}\}\tau_k^2 + (\mathbb{E}\{Y_{i,k}\})^2(\tau_k^2 - \mu_k^2) \\
&\leq \mathbb{E}\{Y_{i,k}\}\tau_k^2,
\end{aligned}$$

and thus $\text{I} = O(n)$ if $\tau_n^2 = O(n)$, $\sum_{k=1}^n \tau_k^2/k = O(n)$, and $\sum_k \tau_k^2/k^2 < \infty$. These conditions hold if $c < 1/2$. We have $\text{I} = o(n^2)$ if $c < 1$.

In II, we have $Y_{i,k} Y_{j,\ell} = 0$ unless the intervals $[i, i + k - 1]$ and $[j, j + \ell - 1]$ are disjoint and precisely one integer apart or nested. For disjoint intervals, we note the independence of $Y_{i,k} Y_{j,\ell}$, $f(\sigma(i, k))$, and $f(\sigma(j, \ell))$, so that

$$\mathbb{E}\{Y_{i,k} f(\sigma(i, k)) Y_{j,\ell} f(\sigma(j, \ell))\} = \mathbb{E}\{Y_{i,k} Y_{j,\ell}\}\mu_k \mu_\ell \ .$$

If none of the intervals contains 1 or n, then a brief argument shows that

$$\mathbb{E}\{Y_{i,k} Y_{j,\ell}\} \leq \frac{4}{(k + \ell + 3)(k + 1)(\ell + 1)} \ .$$

If one interval covers 1 and the other n, then $k + \ell = n - 1$, and $\mathbb{E}\{Y_{i,k} Y_{j,\ell}\} = 1/n$. In the other cases, the expected value is bounded by $2/(k + \ell + 2)(k + 1)$ or $2/(k + \ell + 2)(\ell + 1)$, depending upon which interval covers 1 or n. Thus, the sum in II limited to disjoint intervals is bounded by

$$\begin{aligned}
&n \sum_{k=1}^n \sum_{\ell=1}^n \frac{4\mu_k \mu_\ell}{(k + \ell + 3)(k + 1)(\ell + 1)} + 1 + \sum_{k=1}^n \sum_{\ell=1}^n \frac{4\mu_k \mu_\ell}{(k + \ell + 2)(k + 1)} \\
&\leq 2n \sum_{k=1}^n \sum_{\ell=1}^k \frac{4\mu_k \mu_\ell}{(k + 3)(k + 1)(\ell + 1)} + 1 + 2 \sum_{k=1}^n \sum_{\ell=1}^k \frac{4\mu_k \mu_\ell}{(k + 2)(k + 1)} \ .
\end{aligned}$$

If $\mu_n = O(n^a)$ for $a > 0$, then it is easy to see that the three sums taken together are $O(n^{2a})$.

We next consider nested intervals. For properly nested intervals, with $[i, i+k-1]$ being the bigger one, we have

$$\begin{aligned}\mathbb{E}\{Y_{i,k} f(\sigma(i,k)) Y_{j,\ell} f(\sigma(j,\ell))\} &= \mathbb{E}\{Y_{i,k}\}\mathbb{E}\{f(\sigma(i,k)) Y_{j,\ell} f(\sigma(j,\ell))\} \\ &\le \frac{2M_k \mathbb{E}\{Y_{i,k}\}\mu_\ell}{(\ell+2)(\ell+1)} \ .\end{aligned}$$

Summed over all allowable pairs $(i,k),(j,\ell)$ with the outer interval not covering 1 or n, and noting that in all cases considered, $a < 1$, this yields a quantity not exceeding

$$\begin{aligned}n\sum_{k=1}^{n} k \sum_{\ell=1}^{k} \frac{4M_k\mu_\ell}{(\ell+2)(\ell+1)(k+2)(k+1)} &\le n\sum_{k=1}^{n} M_k O(k^{a-2}) \\ &= \begin{cases} O(n^{b+a}) & \text{if } b+a \neq 1 \ , \\ O(n\log n) & \text{if } b+a = 1. \end{cases}\end{aligned}$$

The contribution of the border effect is of the same order. This is $o(n^2)$ if $a+b<2$. It is $O(n)$ if $a+b \le 1$.

Finally, we consider nested intervals with $i = j$ and $\ell < k$. Then

$$\mathbb{E}\{Y_{i,k} f(\sigma(i,k)) Y_{j,\ell} f(\sigma(j,\ell))\} \le \mathbb{E}\{Y_{i,k}\} M_k \frac{\mu_\ell}{\ell+1} \ .$$

Summed over all appropriate (i,k,ℓ) such that the outer interval does not cover 1 or n, we obtain a bound of

$$n\sum_{k=1}^{n}\sum_{\ell=1}^{k} \frac{2M_k\mu_\ell}{(k+2)(k+1)(\ell+1)} = O(n^{a+b} + \mathbb{1}_{[a+b=1]} n\log n) \ .$$

The border cases do not alter this bound. Thus, the contribution to II for these nested intervals is $o(n^2)$ if $a+b<2$ and is $O(n)$ if $a+b<1$. □

4. A law of large numbers. The estimates of the previous section permit us to obtain a law of large numbers.

THEOREM 1. *Assume that $M_n < \infty$ for all n and that $f \ge 0$. Assume that for some $b \ge c \ge a > 0$, we have $\mu_n = O(n^a)$, $\tau_n = O(n^c)$, and $M_n = O(n^b)$. If $a+b<2$, $c<1$, then*

$$\frac{X_n}{n} \to \mu$$

in probability. If f is a toll function and $M_n = O(n^b)$, then $X_n/n \to \mu$ in probability when $b<1$.

Proof. Note that $a<1$. By Lemma 1, we have $\mathbb{E}\{X_n\}/n \to \mu$. Choose $\epsilon > 0$. By Chebyshev's inequality and Lemma 2,

$$\mathbb{P}\{|X_n - \mathbb{E}\{X_n\}| > \epsilon n\} \le \frac{\mathbb{V}\{X_n\}}{\epsilon^2 n^2} = o(1) \ .$$

Thus, $X_n/n - \mathbb{E}\{X_n\}/n \to 0$ in probability. □

Four examples will illustrate this result.

EXAMPLE 6. *We let f be the indicator function of anything, and note that the law of large numbers holds. For example, let $\mathcal{T}$ be a possibly infinite collection of possible tree patterns, and let X_n count the number of subtrees in a random binary search tree that match a tree from $\mathcal{T}$. Then, as shown below, the law of large numbers holds. There is inherent limitation to $\mathcal{T}$, which, in fact, might be the collection of all trees whose size is a perfect square and whose height is a prime number at the same time. Let X_n be the number of subtrees in a random binary search tree that match a given prefix tree pattern T, with $|T| = k$ fixed.*

THEOREM 2. *For any nonempty tree pattern collection $\mathcal{T}$, we have*

$$\frac{X_n}{n} \to \mu$$

in probability, and $\mathbb{E}\{X_n\}/n \to \mu$, where

$$\mu = \sum_{n=1}^{\infty} \frac{2\mu_n}{(n+2)(n+1)}$$

and μ_n is the probability that a random binary search tree of size n matches an element of $\mathcal{T}$.

Proof. Theorem 1 applies since f is an indicator function. By Lemma 1, we obtain the limit μ for $\mathbb{E}\{X_n\}/n$. □

Note that Theorem 2 remains valid if we replace the phrase "matches an element of $\mathcal{T}$" by the phrase "matches an element of $\mathcal{T}$ at its root," so that $\mathcal{T}$ is a collection of what we called earlier prefix tree patterns.

EXAMPLE 7. *Perhaps more instructive is the example of the sumheight $\mathcal{S}_n$, the sum of the heights of all subtrees in a random binary search tree on n nodes.*

THEOREM 3. *For a random binary search tree, the sumheight satisfies*

$$\frac{\mathcal{S}_n}{\mathbb{E}\{\mathcal{S}_n\}} \to 1$$

in probability. Here

$$\mathbb{E}\{\mathcal{S}_n\} \sim n \sum_{k=1}^{\infty} \frac{2h_k}{(k+2)(k+1)} \ ,$$

where h_k is the expected height of a random binary search tree on k nodes.

Proof. The statement about the expected height follows from Lemma 1 without work. As the height of a subtree of size k is at most $k-1$, we see that we may apply Theorem 1 with $M_k = k-1$. By well-known results (Robson (1979), Pittel (1984), Devroye (1986), (1987)), we have $\mathbb{E}\{H_n^2\} = O(\log^2 n)$, where H_n is the height of a random binary search tree. Thus, we may formally take a and c arbitrarily small but positive, and $b = 1$. □

EXAMPLE 8. *Define $L(u)$ to be the largest number of full levels below u, and let $C(u) = 2^{L(u)+1} - 1$ be the size of that largest full subtree rooted at u. Define*

$$X_n = \sum_u C(u) \ .$$

This parameter measures to some extent the amount of balance in the tree.

THEOREM 4. *For a random binary search tree,*

$$\frac{X_n}{n} \to \mu$$

in probability, and $\mathbb{E}\{X_n\}/n \to \mu$, *where*

$$\mu = \sum_{n=1}^{\infty} \frac{2\mu_n}{(n+2)(n+1)}$$

and μ_n *is the expected size of the largest complete subtree rooted at the root of a random binary search tree of size* n.

Proof. We verify that Theorem 1 and Lemma 1 may be applied with $a = 0.35$, $b = 1$, and $c = 0.35$. Indeed, if H_n is the number of full levels starting at a level below the root in a random binary search tree on n nodes, we know from Devroye (1986) that $L_n/\log n \to \gamma = 0.3711\cdots$ in probability and in the mean. This result does not suffice, as we need to show that $\mathbb{E}\{2^{L_n}\} = O(n^a)$ with $a = 0.35$. But using a representation for tree sizes in terms of products of independent uniform $[0,1]$ random variables $U_1, \ldots, U_n$ (Devroye (1986)) (the tree size for any node at distance k from the root is distributed as $\lfloor \cdots \lfloor \lfloor nU_1 \rfloor U_2 \rfloor \cdots U_k \rfloor$), we see that

$$\begin{aligned}
\mathbb{P}\{L_n \geq k\} &\leq \left(\mathbb{P}\{nU_1 \cdots U_k \geq 1\}\right)^{2^k} \\
&\leq \exp\left(-2^k \mathbb{P}\{ne^{-G_k} < 1\}\right) \\
&\leq \exp\left(-2^k \mathbb{P}\{G_k > \log n\}\right) \\
&\leq \exp\left(-2^k \int_{\log n}^{\infty} y^{k-1}/(k-1)!e^{-y}dy\right) \\
&\leq \exp\left(-2^k (\log n)^{k-1}/(k-1)!n\right) \\
&\leq \exp\left(-2^k (\log n)^{k-1}\sqrt{k}/(k/e)^k e\sqrt{2\pi}n\right) \\
&\qquad \text{(by Stirling's approximation)} \\
&\leq \exp\left(-(2e\log n/k)^k \sqrt{k}/e\sqrt{2\pi}n\log n\right) \\
&= \exp\left(-(2e^{1-1/c}/c)^{c\log n}\sqrt{c}/e\sqrt{2\pi\log n}\right) \\
&\qquad \text{(after setting } k = c\log n\text{)} \\
&\leq \exp\left(-n^{\log\sqrt{4/e}}/e\sqrt{4\pi\log n}\right) \\
&\qquad \text{(by the choice } c = 1/2\text{)}.
\end{aligned}$$

Clearly, then,

$$\mathbb{E}\{2^{L_n}\} \leq n\mathbb{P}\{L_n \geq (1/2)\log n\} + 2^{(1/2)\log n} = o(1) + n^{\log\sqrt{2}} = o(n^{0.35}) \ .$$

We also have $\mathbb{E}\{2^{2L_n}\} = o(n^{0.7})$ by the same argument. Thus, both the conditions of Lemma 1 and Theorem 1 are satisfied and the law of large numbers follows. □

EXAMPLE 9. *Consider* $X_n = \sum_u (N(u))^{0.999}$. *Recall that* $\sum_u N(u)$ *is the number of comparisons in quicksort, plus* n. *Thus,* X_n *is a discounted parameter with respect to the number of quicksort comparisons. Clearly, Theorem* 1 *applies with* $a = b = c = 0.999$, *and thus* $X_n/n \to \mu$ *in probability, and* $\mathbb{E}\{X_n\}/n$ *tends to the same constant* μ. *In a sense, this application is near the limit of the range for Theorem* 1. *For example,*

it is known that with $X_n = \sum_u (N(u))^{1+\epsilon}$*, there is no asymptotic concentration, and thus,* $X_n/g(n)$ *does not converge to a constant for any choice of* $g(n)$*. Also, for* $X_n = \sum_u (N(u))^1$*, the quicksort example, we have* $X_n/2n\log n \to 1$ *in probability (Sedgewick (*1983*)), so that once again Theorem* 1 *is not applicable. Therefore, in a vague sense, the conditions of Theorem* 1 *are nearly best possible, as the theorem applies to* $X_n = \sum_u (N(u))^{1-\epsilon}$ *with* $\epsilon \in (0, 1]$*.*

5. Dependency graph. We will require the notion of a dependency graph for a collection of random variables $(Z_\alpha)_{\alpha\in V}$, where V is a set of vertices. Let the edge set E be such that for all disjoint subsets A and B of V, either there is an edge of E between A and B or there is no edge, and in the latter case, $(Z_\alpha)_{\alpha\in A}$ and $(Z_\alpha)_{\alpha\in B}$ are mutually independent. Clearly, the complete graph is a dependency graph for any set of random variables, but this is useless. One usually takes the minimal graph (V, E) that has the above property, or one tries to keep $|E|$ as small as possible. Note that, necessarily, Z_α and Z_β are independent if $(\alpha, \beta) \notin E$, but to have a dependency graph requires much more than just checking pairwise independence. We call the neighborhood of $N(\alpha)$ of vertex $\alpha \in V$ the collection of vertices β such that $(\alpha, \beta) \in E$ or $\alpha = \beta$. We define the neighborhood $N(\alpha_1, \ldots, \alpha_r)$ as $\cup_{j=1}^r N(\alpha_j)$.

A. Now consider for V the pairs (i, k) with $1 \le i \le n$ and $1 \le k \le n - i + 1$. Let our collection of random variables be the permutations $\sigma(i, k)$, $(i, k) \in V$. Let us connect (i, k) to (j, ℓ) when $i + k \ge j - 1$ and $j + \ell \ge i$. This means that the intervals $(i, i + k - 1)$ and $(j, j + \ell - 1)$ correspond to an edge in E if and only if they overlap or are disjoint and separated by exactly zero or one integer m. We claim that (V, E) is a dependency graph. Indeed, if we consider disjoint subsets A and B of vertices with no edges between them, then these vertices correspond to intervals that are pairwise separated by at least two integers, and thus $(\sigma(i, k))_{(i,k)\in A}$ and $(\sigma(j, \ell))_{(j,\ell)\in B}$ are mutually independent.

B. Consider next the collection of random variables $Y_{i,k}g(k)$. For this collection, we can make a smaller dependency graph. Eliminate all edges from the graph of the previous paragraph if the intervals defined by the endpoints of the edges are properly nested. For example, if $i < j < j + \ell - 1 < i + k$, then the edge between (i, k) and (j, ℓ) is removed. The graph thus obtained is still a dependency graph. This observation repeatedly uses the fact that if one considers a sequence $Z_1, \ldots, Z_n$ of i.i.d. random variables with a uniform $[0, 1]$ distribution, then Z_1, Z_n and the permutation of $Z_2, \ldots, Z_{n-1}$ are all independent. Thus, for properly nested intervals as above, $Y_{i,k}g(k)$ is independent of $Y_{j,\ell}g(\ell)$.

C. A third dependency graph that will be useful is constructed as above when V is restricted to those pairs (i, k) with $1 \le i \le n$, $1 \le k \le n - i + 1$, and, additionally, $k \le K$. Typically, $K = o(n)$, so this will restrict the degree of each vertex in the dependency graph. For example, given any vertex (i, k) in this graph, its neighborhood $N((i, k))$ has cardinality bounded by $(2K+2)K$, because the starting point for a connected interval has at most $2K+2$ choices and the length at most K.

6. Stein's method. Stein's method (Stein (1972)) allows one to deduce a normal limit law for certain sums of random variables while computing only first and second order moments and verifying a certain dependence condition. Many variants have seen the light of day in recent years, and we will simply employ the following version derived in Janson, Łuczak, and Ruciński (2000, Theorem 6.33).

LEMMA 3. *Suppose that $(S_n)_1^\infty$ is a sequence of random variables such that $S_n = \sum_{\alpha \in V_n} Z_{n\alpha}$, where for each n, $\{Z_{n\alpha}\}_\alpha$ is a family of random variables with dependency graph (V_n, E_n). Let $N(.)$ denote the neighborhood of a vertex or vertices. Further suppose that there exist numbers M_n and Q_n such that*

$$\sum_{\alpha \in V_n} \mathbb{E}\{|Z_{n\alpha}|\} \le M_n$$

and for every $\alpha, \alpha' \in V_n$

$$\sum_{\beta \in N(\alpha, \alpha')} \mathbb{E}\{|Z_{n\beta}||Z_{n\alpha}, Z_{n\alpha'}\} \le Q_n \ .$$

Let $\sigma_n^2 = \mathbb{V}\{S_n\}$. Then

$$\frac{S_n - \mathbb{E}\{S_n\}}{\sqrt{\mathbb{V}\{S_n\}}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1)$$

if

$$\lim_{n \to \infty} \frac{M_n Q_n^2}{\sigma_n^3} = 0 \ .$$

Proof. We apply Lemma 3 with the basic collection of random variables $Y_{i,k} f(\sigma(i,k))$, $(i,k) \in V_n$, where V_n is the collection $\{(i,k) : 1 \le i \le n, 1 \le k \le \min(K, n-i+1)\}$. Let E_n be the edges in the dependency graph L_n defined by connecting (i,k) to (j,ℓ) if the respective intervals are overlapping or if the respective intervals are disjoint with zero or one integer separating them. We note that

$$\sum_{(i,k) \in V_n} \mathbb{E}\{Y_{i,k} g(k)\} \le (\nu + o(1))n$$

by computations not unlike those for the mean done earlier, where

$$\nu = \sum_{n=1}^{\infty} \frac{2g(n)}{(n+2)(n+1)} \ .$$

To apply Lemma 3, we note that we may thus take $M_n = O(n)$. We also note that $\sigma_n^2 = \Omega(n)$ by assumption. Define

$$Q_n = \sup_{(i,k),(j,\ell) \in V_n} \sum_{(p,r) \in N((i,k),(j,\ell))} \mathbb{E}\{Y_{p,r} g(r) | Y_{i,k} f(\sigma(i,k)), Y_{j,\ell} f(\sigma(j,\ell))\} \ .$$

Indeed, as g bounds $|f|$, this is all we need to bound. The technical condition in Lemma 3 is satisfied if $Q_n = o(n^{1/4})$. To compute an upper bound for Q_n, we bound as follows:

$$Q_n \le \sup_{(i,k),(j,\ell) \in V_n} |N((i,k),(j,\ell))| g(K).$$

Each of the intervals represented by (i,k) and (j,ℓ) has length at most K. Clearly, $(p,r) \in N((i,k),(j,\ell))$ means that both p and $p+r-1$ must be in these intervals or

within the $K+1$ neighbors of them. Each of p and r has thus at most $3K+2$ choices, so that $|N((i,k),(j,\ell))| \le (3K+2)^2$. Thus,

$$Q_n \le (3K+2)^2 g(K)$$

from which Lemma 3 follows without further work. □

A simple example counts the number of subtrees in a random binary search tree that match one of a given collection of tree patterns (these are "terminal matches" at the bottom of the tree), where each pattern is of size $\le K$, where K may depend upon n. As f is an indicator function, we may take $g \equiv 1$. Let X_n denote the number of matches.

LEMMA 4. *Let X_n be the number of matches of a tree pattern in a collection of tree patterns depending arbitrarily on n, as long as, within the collection, the maximal tree size is $K = K(n)$. If*

$$\lim_{n\to\infty} \frac{\mathbb{E}\{X_n\}K^2}{(\mathbb{V}\{X_n\})^{3/2}} = 0 \ ,$$

then

$$\frac{X_n - \mathbb{E}\{X_n\}}{\sqrt{\mathbb{V}\{X_n\}}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1) \ .$$

Proof. Follow the proof of Lemma 3, but do not use the estimate $\sigma_n^2 = \Omega(n)$. □

Note that the above result remains true even if the collection of patterns itself is a function of n, changing in cardinality and in membership with n, within the condition imposed on K. This result extends the central limit laws of Devroye (1991), where K had to remain fixed. Indeed, the technical condition of Lemma 4 becomes $\mathbb{V}\{X_n\}/\mathbb{E}^{2/3}\{X_n\} \to \infty$. Note in this respect that for K fixed, and the collection of tree patterns nonempty for all n, we have $\mathbb{V}\{X_n\} = \Theta(n)$ and $\mathbb{E}\{X_n\} = \Theta(n)$, facts that are easy to verify.

7. Sums of indicator functions. In this section, we take a simple example in which

$$X_n = \sum_u \mathbb{1}_{[S(u)\in A_n]} \ ,$$

where A_n is a nonempty collection of permutations of length k, with k possibly depending upon n. We denote $p_{n,k} = |A_n|/k!$, the probability that a randomly picked permutation of length k is in the collection A_n. Particular examples include sets A_n that correspond to a particular tree pattern, in which case X_n counts the number of occurrences of a given tree pattern of size k (a "terminal pattern") in a random binary search tree. The interest here is in the case of varying k. As we will see below, for a central limit law, k has to be severely restricted.

Our main result is the following theorem.

THEOREM 5. *We have*

$$\mathbb{E}\{X_n\} = \frac{2np_{n,k}}{(k+2)(k+1)} + O(1)$$

regardless of how k varies with n. If $k = o(\log n/\log\log n)$, then $\mathbb{E}\{X_n\} \to \infty$, $X_n/\mathbb{E}\{X_n\} \to 1$ in probability, and

$$\frac{X_n - \mathbb{E}\{X_n\}}{\sqrt{\mathbb{V}\{X_n\}}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1) \ .$$

Proof. Observe that

$$X_n = \sum_{i=1}^{n-k+1} Y_{i,k} Z_i \ ,$$

where $Z_i = \mathbb{1}_{[\sigma(i,k)\in A_n]}$. Thus,

$$\mathbb{E}\{X_n\} = \sum_{i=2}^{n-k} \frac{2}{(k+2)(k+1)}\mathbb{E}\{Z_1\} + 2 \times \frac{1}{k+1}\mathbb{E}\{Z_1\} = \frac{2(n-k-1)p_{n,k}}{(k+2)(k+1)} + \frac{2p_{n,k}}{k+1} \ .$$

This proves the first part of the theorem.

The computation of the variance is slightly more involved. However, it is simplified by considering the variance of

$$Y_n = \sum_{i=2}^{n-k} Y_{i,k} Z_i$$

and noting that $|X_n - Y_n| \le 2$. This eliminates the border effect. We note that $Y_{i,k}Y_{j,k} = 0$ if $i < j \le i + k$. Thus,

$$\begin{aligned}
\mathbb{E}\{Y_n^2\} &= \sum_{i=2}^{n-k} \mathbb{E}\{Y_{i,k}Z_i\} + 2 \sum_{2\le i<j\le n-k} \mathbb{E}\{Y_{i,k}Z_iY_{j,k}Z_j\} \\
&= \mathbb{E}\{Y_n\} + 2 \sum_{2\le i,i+k+1\le n-k} \mathbb{E}\{Y_{i,k}Z_iY_{i+k+1,k}Z_{i+k+1}\} \\
&\qquad + 2 \sum_{2\le i,i+k+1<j\le n-k} \mathbb{E}\{Y_{i,k}Z_i\}\mathbb{E}\{Y_{j,k}Z_j\} \\
&= (n-k-1)\beta + 2(n-2k-2)\alpha + (n-2k)^2\beta^2 + (10k+6-5n)\beta^2,
\end{aligned}$$

where $\alpha = \mathbb{E}\{Y_{2,k}Z_2Y_{3+k,k}Z_{3+k}\}$ and $\beta = \mathbb{E}\{Y_{2,k}Z_2\}$. Also,

$$(\mathbb{E}\{Y_n\})^2 = ((n-k-1)\beta)^2 \ .$$

Thus,

$$\begin{aligned}
\mathbb{V}\{Y_n\} &= 2(n-2k-2)\alpha + (n-k-1)\beta \\
&\qquad + \left((n-2k)^2 - (n-k-1)^2 + (10k+6-5n)\right)\beta^2 \\
&= n\left(2\alpha + \beta - (2k+3)\beta^2\right) + O(k\alpha + k\beta + k^2\beta^2) \ .
\end{aligned}$$

We note that $\beta = 2p_{n,k}/(k+2)(k+1)$. To compute α, let A, B, C be the minimal values among $U_1, \ldots, U_{k+1}$, U_{k+2}, and $U_{k+3}, \ldots, U_{2k+3}$, respectively. Clearly,

$$\alpha = p_{n,k}^2 \mathbb{E}\{Y_{2,k}Y_{3+k,k}\}.$$

Considering all six permutations of A, B, C separately, one may compute the latter expected value as

$$\frac{2}{2k+3}\frac{1}{2k+2}\frac{1}{k+1}+\frac{1}{2k+3}\frac{1}{k+1}\frac{1}{k+1}+\frac{2}{2k+3}\frac{1}{2k+2}\frac{1}{2k+1}$$
$$=\frac{5k+3}{(2k+3)(2k+1)(k+1)^2}.$$

Thus,

$$\alpha=\frac{(5k+3)p_{n,k}^2}{(2k+3)(2k+1)(k+1)^2}.$$

We have

$$\mathbb{V}\{Y_n\}$$
$$=n\left(p_{n,k}^2\frac{10k+6}{(2k+3)(2k+1)(k+1)^2}-p_{n,k}^2\frac{8k+12}{(k+2)^2(k+1)^2}+p_{n,k}\frac{2}{(k+2)(k+1)}\right)$$
$$+O(p_{n,k}/k).$$

Note that regardless of the value of $p_{n,k}$, the coefficient of n is strictly positive. Indeed, the coefficient is at least

$$p_{n,k}^2\left(\frac{(10k+6)(k+2)^2-(8k-8)(2k+3)(2k+1)+2(2k+3)(2k+1)(k+2)(k+1)}{(k+2)^2(k+1)^2(2k+3)(2k+1)}\right)$$
$$=p_{n,k}^2\left(\frac{8k^4+18k^3+4k^2-6k}{(k+2)^2(k+1)^2(2k+3)(2k+1)}\right).$$

Thus, there exist universal constants $c_1, c_2, c_3 > 0$ such that

$$\mathbb{V}\{Y_n\}\geq c_1 np_{n,k}^2/k^2-c_2p_{n,k}/k$$

and

$$\mathbb{V}\{Y_n\}\leq c_3 np_{n,k}/k^2\ .$$

We have $Y_n/\mathbb{E}\{Y_n\}\to 1$ in probability if $\mathbb{V}\{Y_n\}=o(\mathbb{E}^2\{Y_n\})$, i.e., if $k=o(\sqrt{np_{n,k}})$. Using $p_{n,k}\geq 1/k!$, we note that this condition holds if $k=o(\log n/\log\log n)$.

Finally, we turn to the normal limit law and note that

$$\frac{Y_n-\mathbb{E}\{Y_n\}}{\sqrt{\mathbb{V}\{Y_n\}}}\xrightarrow{\mathcal{L}}\mathcal{N}(0,1)$$

if (see Lemma 4)

$$\lim_{n\to\infty}\frac{k^2\mathbb{E}\{Y_n\}}{(\mathbb{V}\{Y_n\})^{3/2}}=0\ .$$

This holds if

$$\lim_{n\to\infty}\frac{np_{n,k}}{n^{3/2}p_{n,k}^3/k^3}=0$$

and $np_{n,k}/k\to\infty$. Both conditions are satisfied if $k=o(\log n/\log\log n)$. □

The limitation on k in Theorem 5 cannot be lifted without further conditions: indeed, if we consider as a tree pattern the tree that consists of a right branch of length k only, then $\mathbb{E}\{X_n\}\to 0$ if $k>(1+\epsilon)\log n/\log\log n$ for any given fixed $\epsilon>0$. As X_n is integer-valued, no meaningul limit laws can exist in such cases.

8. Notes on the variance. For toll functions, that is, functions f such that $f(\sigma) = g(|\sigma|)$ for some function g, we need the following lemma.

LEMMA 5. *Define* $X_n = \sum_u g(|S(u)|)$ *and* $X_{n,K} = \sum_u g(|S(u)|)\mathbb{1}_{[|S(u)|\le K]}$ *for a random binary search tree. The following statements are equivalent:*

A. $\mathbb{V}\{X_{n,K}\} = \Omega(n)$ *for any* K *with* $K \to \infty$ *as* $n \to \infty$.

B. $\mathbb{V}\{X_n\} = \Omega(n)$.

C. $\mathbb{V}\{X_k\} > 0$ *for some* $k > 0$.

D. *The function* g *is not constant on* $\{1, 2, \ldots\}$.

Proof. D implies C. Indeed, let k be the first integer at least equal to 2 such that

$$g(k) \neq g(k-1) = \cdots = g(1) \ .$$

For the integers up to k, we have the representation $g(x) = c + d\mathbb{1}_{[x=k]}$ with $d \neq 0$. We have $X_i = ic$, $i < k$, $X_k = kc + d$, and $X_{k+1} = f(k+1) + kc + dN_k$, where N_k is the number of nodes for which $|S(u)| = k$. Note that $N_k = 1$ with probability $2/(k+1)$ and 0 otherwise. Thus,

$$\mathbb{V}\{X_{k+1}\} = d^2 \times \frac{2(k-1)}{(k+1)^2} > 0 \ .$$

C implies A. For two random variables W, Y, we have $\mathbb{V}\{W\} = \mathbb{E}\{\mathbb{V}\{W|Y\}\} + \mathbb{V}\{\mathbb{E}\{W|Y\}\}$. Thus,

$$\mathbb{V}\{X_{n,K}\} \ge \mathbb{E}\{\mathbb{V}\{X_{n,K}|\mathcal{F}_k\}\},$$

where $\mathcal{F}_k$ is defined as follows. Identify in the permutation that defines the tree all nodes u for which $|S(u)| = k$. By construction, each $S(u)$ corresponds to an interval of the original random permutation (of length n). Let $\mathcal{F}_k$ be all elements of the original random permutation except those corresponding to the intervals representing $S(u)$ with $|S(u)| = k$. By the conditional independence of the various $S(u)$'s of size k (none can overlap), and defining $N_k = \sum_u \mathbb{1}_{[|S(u)|=k]}$, we have for n so large that $K \ge k$,

$$\begin{aligned}
\mathbb{V}\{X_{n,K}\} &\ge \mathbb{E}\left\{ \sum_{u:|S(u)|=k} \mathbb{V}\{X_{k,K}|\mathcal{F}_k\} \right\} \\
&= \mathbb{E}\left\{N_k \mathbb{V}\{X_{k,K}\}\right\} \\
&= \mathbb{E}\{N_k\}\mathbb{V}\{X_{k,K}\} \\
&= \mathbb{E}\{N_k\}\mathbb{V}\{X_k\} \\
&\sim \frac{2n\mathbb{V}\{X_k\}}{(k+2)(k+1)} \ .
\end{aligned}$$

Thus, the lower bound follows if $\mathbb{V}\{X_k\} > 0$ for some k.

A implies B. Just take $K = n$.

B implies D. If g is constant on the positive integers, then $\mathbb{V}\{X_k\} = 0$ for all k. □

For general functions f on the set of all permutations, it is always possible to have $X_n = 0$ (and thus $\mathbb{V}\{X_n\} = 0$) along a subsequence for n. Assume that all values for $f(\sigma)$ are given, with $|\sigma| < n$. Let σ be of size n. Define $f(\sigma)$ such that $X_n = \sum_u f(S(u)) = 0$. One can even construct examples in which f is integer-valued and $f(\sigma) = |\sigma| = n$ while, for smaller permutations σ', the values $f(\sigma')$ are quite

arbitrary as long as they are taken from $\{0, 1, 2, \ldots, |\sigma'|\}$. In the latter case, we have $X_n \equiv n$ for that particular n. So, faced with these pesky examples, we will develop the sequel with the condition $\mathbb{V}\{X_n\} = \Omega(n)$ thrown in. The reader should be warned that this condition needs rigorous verification in each application that does not deal with a toll function.

9. Sums of functions of sizes of subtrees. In this section, we consider two types of random variables,

$$X_n = \sum_u g(|S(u)|)$$

and

$$X_{n,K} = \sum_{u:|S(u)|\le K} g(|S(u)|) \ ,$$

where $K = K(n) \le n$ is a sequence of positive numbers. Define $G(n) = \max_{1\le i\le n} |g(i)|$.

LEMMA 6. *Assume that g is not constant on $\{1, 2, \ldots\}$. If $K \to \infty$, and $G(K)\log^2 K = o(n^{1/4})$, then*

$$\frac{X_{n,K} - \mathbb{E}\{X_{n,K}\}}{\sqrt{\mathbb{V}\{X_{n,K}\}}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1) \ .$$

Proof of Lemma 6. We apply Lemma 3 with the basic collection of random variables $Y_{i,k}g(k)$, $(i,k) \in V_n$, where V_n is the collection $\{(i,k) : 1 \le i \le n, 1 \le k \le \min(K, n-i+1)\}$. Let E_n be the edges in the dependency graph L_n defined by connecting (i,k) to (j,ℓ) if the respective intervals are overlapping without being properly nested, or if the respective intervals are disjoint with zero or one integers separating them. (Note that the dependency graph is thus considerably smaller than in the proof of Theorem 5.) We note that

$$\sum_{(i,k)\in V_n} \mathbb{E}\{Y_{i,k}|g(k)|\} \le (\nu + o(1))n$$

by computations not unlike those for the mean done earlier, where

$$\nu = \sum_{n=1}^{\infty} \frac{2|g(n)|}{(n+2)(n+1)} \ .$$

To apply Lemma 3, we note that we may thus take $M_n = O(n)$. We also note that $\sigma_n^2 = \Omega(n)$, by Lemma 5, since $K \to \infty$ and g is not constant on $\{1, 2, \ldots\}$. It thus suffices to show that $Q_n = o(n^{1/4})$. Note that conditioning on $Y_{i,k}g(k)$ is equivalent to conditioning on $Y_{i,k}$. Thus, we may bound Q_n by

$$Q_n \le G(K) \sup_{(i,k),(j,\ell)\in V_n} \sum_{(p,r)\in N((i,k),(j,\ell))} \mathbb{E}\{Y_{p,r}|Y_{i,k}, Y_{j,\ell}\} \ .$$

We show that sum above is uniformly bounded over all choices of $(i,k),(j,\ell)$ by $O(\log^2 K)$.

Consider the set $S = \{0, 1, \ldots, n, n+1\}$ and mark $0, n+1, i-1, i+k, j-1, j+\ell$ (where duplications may occur). The last four marked points are neighbors of

the intervals represented by (i,k) and (j,ℓ). Also mark all integers in S that are neighbors of these marked numbers. The total number of marked places does not exceed $3 \times 4 + 2 \times 2 = 16$. The set S, when traversed from small to large, can be described by consecutive intervals of marked and unmarked integers. The number of unmarked integer intervals is at most five. We call these intervals $H_1, \ldots, H_5$, from left to right, with some of these possibly empty. Set $H = \cup_i H_i$. Define $H^c = S - H$. Consider $Y_{p,r}$ for $r \le K$ fixed. Let $s = p + r - 1$ be the endpoint of the interval on which $Y_{p,r}$ sits. Note that $Y_{p,r}$ depends upon $\{U_i\}_{p-1 \le i \le s+1}$. We note four situations:

A. If $p, s \in H_i$ for a given i, then $Y_{p,r}$ clearly is independent of $Y_{i,k}, Y_{j,\ell}$. In fact, then, $(p,r) \notin N((i,k),(j,\ell))$.

B. If $p, s \in H^c$, then we bound $\mathbb{E}\{Y_{p,r}|Y_{i,k}, Y_{j,\ell}\}$ by one.

C. If p or s is in H^c and the other endpoint is in H_i, then we bound as follows:

$$\mathbb{E}\{Y_{p,r}|Y_{i,k}, Y_{j,\ell}\} \le \frac{1}{1 + |H_i \cap \{p, \ldots, s\}|}$$

because we can only be sure about the i.i.d. nature of the U_i's in $H_i \cap \{p, \ldots, s\}$ together with the two immediate neighbors of this set.

D. If $p \in H_i$, $s \in H_j$, $i < j$, then we argue as in case C twice, and obtain the following bound:

$$\mathbb{E}\{Y_{p,r}|Y_{i,k}, Y_{j,\ell}\} \le \frac{1}{1 + |H_i \cap \{p, \ldots, s\}|} \times \frac{1}{1 + |H_j \cap \{p, \ldots, s\}|} \ .$$

The above considerations permit us to obtain a bound for Q_n by summing over all $(p,r) \in N((i,k),(j,\ell))$. The sum for all cases A is zero. The sum for case B is at most $16^2 = 256$. The sum over all (p,r) as in C is at most

$$2 \times 16 \times 5 \times \sum_{r=1}^{K} \frac{1}{r+1} \le 160 \log(K+1) \ .$$

Finally, the sum over all (p,r) described by D is at most

$$\binom{5}{2} \left(\sum_{r=1}^{K} \frac{1}{r+1} \right)^2 \le 10 \log^2(K+1) \ .$$

The grand total is $O(\log^2 K)$, as required. This concludes the proof of Lemma 6. □

COROLLARY 1. *As $K \le n$, we deduce that for $G(n) = o(n^{1/4}/\log^2 n)$,*

$$\frac{X_n - \mathbb{E}\{X_n\}}{\sqrt{\mathbb{V}\{X_n\}}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1) \ .$$

This result will be slightly improved in Theorem 6 below.

COROLLARY 2. *A sufficient condition for Lemma 6 is $K = O(n^a)$ for some $0 \le a \le 1$, and $G(n) = o(n^{1/4a}/\log^2 n)$. Other sufficient conditions include either $K = O(1)$ or $K = O(\log n)$, $G(n) = O(\exp(\epsilon n))$ for all $\epsilon > 0$.*

THEOREM 6. *Assume that g is not constant on $\{1, 2, \ldots\}$. If $G(n) = o(n^{1/3}/\log^2 n)$, then*

$$\frac{X_n - \mathbb{E}\{X_n\}}{\sqrt{\mathbb{V}\{X_n\}}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1) \ .$$

Proof. Theorem 6 follows directly from Lemma 6 if we can prove the following facts with $K = \lfloor n^{3/4} \rfloor$:

A. For all $\epsilon > 0$, $\lim_{n\to\infty} \mathbb{P}\{X_n - X_{n,K} \geq \epsilon\sqrt{n}\} = 0$. A sufficient condition is $\mathbb{E}\{X_n - X_{n,K}\} = o(\sqrt{n})$.

B. $\liminf_{n\to\infty} \mathbb{V}\{X_n\}/n > 0$.

C. $\mathbb{V}\{X_{n,K}\} \sim \mathbb{V}\{X_n\}$.

For part A, note the following:

$$\begin{aligned}
|\mathbb{E}\{X_n - X_{n,K}\}| &= \left| \sum_{(i,k):1\leq i\leq n, K<k\leq n-i+1} \mathbb{E}\{Y_{i,k}\} g(k) \right| \\
&\leq G(n) + 2 \sum_{K<k\leq n-1} \mathbb{E}\{Y_{1,k}\} G(k) + \sum_{(i,k):2\leq i\leq n, K<k\leq n-i} \mathbb{E}\{Y_{i,k}\} G(k) \\
&\leq G(n) + 2 \sum_{K<k\leq n-1} \frac{G(k)}{k+1} + \sum_{(i,k):2\leq i\leq n, K<k\leq n-i} \frac{2G(k)}{(k+2)(k+1)} \\
&\leq o(n^{1/3}) + \sum_{k=K+1}^{n} \sum_{i=2}^{n-k} \frac{2G(k)}{(k+2)(k+1)} \\
&\leq o(n^{1/3}) + n \sum_{k=K+1}^{n} \frac{2G(k)}{(k+2)(k+1)} \\
&= o(n^{1/2})
\end{aligned}$$

by our choice of K.

Part B is immediate from Lemma 5.

For part C, set

$$X_n = X_{n,K} + W_{n,K}$$

and note that

$$\mathbb{V}\{X_n\} = \mathbb{V}\{X_{n,K}\} + \mathbb{V}\{W_{n,K}\} + 2\mathbb{E}\{(X_{n,K} - \mathbb{E}\{X_{n,K}\})(W_{n,K} - \mathbb{E}\{W_{n,K}\})\} \ .$$

We have from Lemmas 5 and 2, $\mathbb{V}\{X_{n,K}\} = \Theta(n)$. We will show that $\mathbb{V}\{W_{n,K}\} = o(n)$. By the Cauchy–Schwarz inequality,

$$\mathbb{E}\{(X_{n,K} - \mathbb{E}\{X_{n,K}\})(W_{n,K} - \mathbb{E}\{W_{n,K}\})\} = o\left(\sqrt{n}\sqrt{\mathbb{V}\{X_{n,K}\}}\right)$$

so that

$$\frac{\mathbb{V}\{X_n\}}{\mathbb{V}\{X_{n,K}\}} = 1 + o(1) + o\left(\frac{\sqrt{n}}{\sqrt{\mathbb{V}\{X_{n,K}\}}}\right) = 1 + o(1) \ .$$

We now show $\mathbb{V}\{W_{n,K}\} = o(n)$. Let $V_n = \{(i,k) : 1 \leq i \leq n, 1 \leq k \leq \min(K, n - i + 1)\}$ be the vertex set and let E_n be the edge set for the dependency graph for the random variables $Y_{i,k}g(k)$. That is, two pairs are connected by an edge if their intervals are not properly nested and are either overlapping or disjoint with at most one integer between them. Then

$$\mathbb{V}\{W_{n,K}\} = \sum_{(i,k)\in V_n} \mathbb{V}\{Y_{i,k}g(k)\} + \sum_{((i,k),(j,\ell))\in E_n} g(k)g(\ell) \left(\mathbb{E}\{Y_{i,k}Y_{j,\ell}\} - \mathbb{E}\{Y_{i,k}\}\mathbb{E}\{Y_{j,\ell}\}\right).$$

The first sum on the right-hand side is bounded by

$$\sum_{(i,k)\in V_n} g^2(k)\mathbb{E}\{Y_{i,k}\} = o(n^{2/3}) + n\sum_{k=K}^{\infty} g^2(k)/k^2 = o(n^{2/3}) + n \times o(K^{-1/3}) = o(n^{3/4}) \ .$$

Consider a fixed edge $((i,k),(j,\ell)) \in E_n$. Note that if the intervals for (i,k) and (j,ℓ) properly overlap without being nested, then $Y_{i,k}Y_{j,\ell} = 0$. The same is true if they are directly adjacent. So, if $((i,k),(j,\ell)) \in E_n$, the product $Y_{i,k}Y_{j,\ell}$ is nonzero only if they are nested and have one coinciding endpoint or if the intervals are separated by precisely one integer. Thus, for the latter intervals with, say, $1 < i \le i+k-1 = j-2 < j \le j+\ell-1 < n$,

$$\mathbb{E}\{Y_{i,k}Y_{j,\ell}\} \le \frac{2}{(\ell+2)(\ell+1)} \times \frac{1}{k+1} \ .$$

For the intervals aligned at i with, say, $1 < i = j \le i+k-1 < j+\ell-1 < n$, we have

$$\mathbb{E}\{Y_{i,k}Y_{j,\ell}\} \le \frac{2}{(\ell+2)(\ell+1)} \times \frac{1}{k+1} \ .$$

The last two bounds are also valid with ℓ and k interchanged. Thus,

$$\begin{aligned}
&\sum_{((i,k),(j,\ell))\in E_n} g(k)g(\ell)\mathbb{E}\{Y_{i,k}Y_{j,\ell}\} \\
&\qquad \le 4G(n) \sum_{(j,\ell)\in V_n} G(\ell)\mathbb{E}\{Y_{j,\ell}\} + \sum_{((i,k),(j,\ell))\in E_n} \frac{2G(k)G(\ell)}{\ell^2 k} \\
&\qquad \le o(n^{1/3}) \sum_{(j,\ell)\in V_n} G(\ell)\mathbb{E}\{Y_{j,\ell}\} + \sum_{((i,k),(i,\ell))\in E_n} \frac{2G(k)G(\ell)}{\ell^2 k} \\
&\qquad\quad + \sum_{((i,k),(i+k+1,\ell))\in E_n} \frac{2G(k)G(\ell)}{\ell^2 k} + \sum_{((j+\ell+1,k),(j,\ell))\in E_n} \frac{2G(k)G(\ell)}{\ell^2 k} \\
&\qquad = \mathrm{I} + \mathrm{II} + \mathrm{III} + \mathrm{IV}.
\end{aligned}$$

First of all,

$$\begin{aligned}
I &\le o(n^{1/3})n \sum_{\ell=K}^{n} \frac{2G(\ell)}{(\ell+2)(\ell+1)} + o(n^{1/3}) \sum_{\ell=K}^{n} \frac{G(\ell)}{\ell+1} \\
&\le \frac{o(n^{4/3}G(n))}{K+1} + o(n^{2/3}) \\
&= o(n^{11/12}).
\end{aligned}$$

Now, $\sum_{k\le\ell} G(k)/k = o(\ell^{1/3})$. Thus, using a symmetry argument,

$$\begin{aligned}
\mathrm{II} &= 2 \sum_{((i,k),(i,\ell))\in E_n, k<\ell} \frac{2G(k)G(\ell)}{\ell^2 k} \le O(n) \sum_{\ell=K}^{n} \frac{o(\ell^{1/3})G(\ell)}{\ell^2} \\
&= O(n) \times o(K^{-1/3}) = o(n^{3/4}).
\end{aligned}$$

Also,

$$\mathrm{III} \le O(n) \sum_{k=K}^{n} G(k)/k \times \sum_{\ell=K}^{n} G(\ell)/\ell^2 = O(n)o(n^{1/3})o(K^{-2/3}) = o(n^{5/6}).$$

Similarly, IV $= o(n^{5/6})$. We conclude that $\mathbb{V}\{W_{n,K}\} = o(n^{11/12})$, which is more than was needed. This concludes the proof of Theorem 6. □

REMARK. Using the contraction method and the method of moments, Hwang and Neininger (2001) showed that the central limit result of Theorem 6 holds with $G(n) = O(n^a)$, $a \leq 1/2$. Our result is weaker, but requires fewer analytic computations.

10. Bibliographic remarks. Central limit theorems for slightly dependent random variables have been obtained by Brown (1971), Dvoretzky (1972), McLeish (1974), Ibragimov (1975), Chen (1978), Hall and Heyde (1980), and Bradley (1981), to name just a few. Stein's method (our Lemma 3, essentially) is one of the central limit theorems that is better equipped to deal with cases of considerable dependence.

Stein's method offers short and intuitive proofs, but other methods may offer attractive alternatives. Hwang and Neininger (2001) are tackling the analysis of random variables of our type by the moment and contraction methods. The ranges of application of the results are not nested; in some situations, Stein's method is more useful, while in others the moment and contraction methods are preferable.

The limit law for L_n, the number of leaves in a random binary search tree, was obtained by Devroye (1991) (see also Mahmoud (1986)):

$$\frac{L_n - \mathbb{E}\{L_n\}}{\sqrt{\mathbb{V}\{L_n\}}} \xrightarrow{\mathcal{L}} \mathcal{N}(0,1) \ .$$

Equivalently, $(L_n - n/3)/\sqrt{n} \xrightarrow{\mathcal{L}} \mathcal{N}(0, 2/45)$. That paper deals with general sums $\sum_i f(\sigma(i,k))$ for k fixed and finite, without regarding the fact that permutations correspond to subtrees of the random binary search tree. As L_n corresponds in our setting to the toll function $\mathbb{1}_{[|\sigma|=1]}$, the above limit law follows easily from Lemma 4, Theorem 5, or Theorem 6. If W_n is the number of nodes with just a right subtree (which occurs at the ith node if and only if $U_i < U_{i+1}$), then it is easy to see that $f(\sigma(i,k)) = \mathbb{1}_{[U_i<U_{i+1}]}$ for all values of k, and this is clearly not covered by Lemma 4. A relatively easy extension would handle it, but we will not be concerned with that here. The result $(W_n - n/2)/\sqrt{n} \xrightarrow{\mathcal{L}} \mathcal{N}(0, 1/12)$ (Devroye (1991)) is thus not an immediate corollary of the present results.

Aldous (1991) showed that the number $V_{k,n}$ of subtrees of size precisely k in a random binary search tree is in probability asymptotic to $2/(k+2)(k+1)$. Devroye showed that $(V_{k,n} - 2n/(k+2)(k+1))/\sqrt{n}$ tends in law to a normal $(0, c_k)$ random variable where c_k is explicitly defined. The latter result also follows from the present paper if we take as toll function $f(\sigma) = \mathbb{1}_{[|\sigma|=k]}$.

Recently, there has been some interest in the logarithmic toll function $f(\sigma) = \log|\sigma|$ (Grabner and Prodinger (2001)) and the harmonic toll function $f(\sigma) = \sum_{i=1}^{|\sigma|} 1/i$ (Panholzer and Prodinger (2001)). The authors in these papers are mainly concerned with precise first and second moment asymptotics. Fill (1996) obtained the central limit theorem for the case $f(\sigma) = \log|\sigma|$. Clearly, these examples fall entirely within the conditions of Lemma 4 or Theorem 6, with some room to spare.

Flajolet, Gourdon, and Martinez (1997) obtained a normal limit law for the number of subtrees in a random binary search tree with fixed finite tree pattern. Clearly, this is a case in which (the indicator function) $f(\sigma)$ depends on σ in an intricate way, but $f = 0$ unless $|\sigma|$ equals the size of the tree pattern. The situation is covered by the law of large numbers of Theorem 2 and the central limit result of Theorem 5. Theorem 5 even allows tree patterns that change with n.

Acknowledgment. We appreciate the comments and improvements suggested to us by Hsien-Kuei Hwang, Ralph Neininger, and Ebrahim Mal-Alla.

REFERENCES

D. Aldous (1991), *Asymptotic fringe distributions for general families of random trees,* Ann. Appl. Probab., 1, pp. 228–266.

R. C. Bradley (1981), *Central limit theorems under weak dependence,* J. Multivariate Anal., 11, pp. 1–16.

B. M. Brown (1971), *Martingale central limit theorems,* Annals of Mathematical Statistics, 42, pp. 59–66.

L. H. Y. Chen (1978), *Two central limit problems for dependent random variables,* Z. Wahrsch. Verw. Gebiete, 43, pp. 223–243.

L. Devroye (1986), *A note on the height of binary search trees,* J. ACM, 33, pp. 489–498.

L. Devroye (1987), *Branching processes in the analysis of the heights of trees, Acta Inform.,* 24, pp. 277–298.

L. Devroye (1988), *Applications of the theory of records in the study of random trees,* Acta Inform., 26, pp. 123–130.

L. Devroye (1991), *Limit laws for local counters in random binary search trees,* Random Structures Algorithms, 2, pp. 303–316.

A. Dvoretzky (1972), *Central limit theorems for dependent random variables,* in Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability, University of California Press, Berkeley, CA, 1972, pp. 513–535.

J. A. Fill (1996), *On the distribution of binary search trees under the random permutation model,* Random Structures Algorithms, 8, pp. 1–25.

P. Flajolet, X. Gourdon, and C. Martinez (1997), *Patterns in random binary search trees,* Random Structures Algorithms, 11, pp. 223–244.

G. H. Gonnet (1984), *Handbook of Algorithms and Data Structures*, Addison–Wesley, Reading, MA.

P. Grabner and H. Prodinger (2001), *Sorting algorithms for broadcast communications: Mathematical analysis,* Theoret. Comput. Sci., to appear.

P. Hall and C. C. Heyde (1980), *Martingale Limit Theory and Its Applications*, Academic Press, New York.

W. Hoeffding and H. Robbins (1949), *The central limit theorem for dependent random variables,* Annals of Mathematical Statistics, 20, pp. 773–780.

H.-K. Hwang and R. Neininger (2001), *Phase Change of Limit Laws in Quicksort Recurrence under Varying Toll Functions,* Technical report, McGill University, Montreal.

I. A. Ibragimov (1975), *A note on the central limit theorem for dependent random variables,* Theory Probab. Appl., 20, pp. 135–141.

S. Janson, T. Łuczak, and A. Ruciński (2000), *Random Graphs*, John Wiley, New York.

D. E. Knuth (1973), *The Art of Computer Programming,* Addison–Wesley, Reading, MA.

D. E. Knuth and A. Schonhage (1978), *The expected linearity of a simple equivalence algorithm,* Theoret. Comput. Sci., 6, pp. 281–315.

W. C. Lynch (1965), *More combinatorial problems on certain trees,* Computer J., 7, pp. 299–302.

H. Mahmoud and B. Pittel (1984), *On the most probable shape of a search tree grown from a random permutation,* SIAM J. Algebraic Discrete Methods, 5, pp. 69–81.

H. M. Mahmoud (1986), *The expected distribution of degrees in random binary search trees,* Computer J., 29, pp. 36–37.

H. M. Mahmoud (1992), *Evolution of Random Search Trees*, John Wiley, New York.

D. L. McLeish (1974), *Dependent central limit theorems and invariance principles,* Ann. Probab., 2, pp. 620–628.

A. Panholzer and H. Prodinger (2001), *Binary search tree recursions with harmonic toll functions,* J. Comput. Appl. Math., to appear.

B. Pittel (1984), *On growing random binary trees,* J. Math. Anal. Appl., 103, pp. 461–480.

J. M. Robson (1979), *The height of binary search trees,* Austral. Comput. J., 11, pp. 151–153.

R. Sedgewick (1983), *Mathematical analysis of combinatorial algorithms,* in Probability Theory and Computer Science, G. Louchard and G. Latouche, eds., Academic Press, London, pp. 123–205.

C. Stein (1972), *A bound for the error in the normal approximation to the distribution of a sum of dependent random variables,* Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability, University of California Press, Berkeley, CA, pp. 583–602.

J. S. Vitter and P. Flajolet (1990), *Average-case analysis of algorithms and data structures,* in Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity, J. van Leeuwen, ed., Elsevier, Amsterdam, pp. 431–524.

SIAM J. COMPUT.
Vol. 32, No. 1, pp. 172–195

APPROXIMATING THE DOMATIC NUMBER*

URIEL FEIGE†, MAGNÚS M. HALLDÓRSSON‡, GUY KORTSARZ§, AND ARAVIND SRINIVASAN¶

Abstract. A set of vertices in a graph is a dominating set if every vertex outside the set has a neighbor in the set. The domatic number problem is that of partitioning the vertices of a graph into the maximum number of disjoint dominating sets. Let n denote the number of vertices, δ the minimum degree, and Δ the maximum degree.

We show that every graph has a domatic partition with $(1 - o(1))(\delta + 1)/\ln n$ dominating sets and, moreover, that such a domatic partition can be found in polynomial-time. This implies a $(1 + o(1))\ln n$-approximation algorithm for domatic number, since the domatic number is always at most $\delta+1$. We also show this to be essentially best possible. Namely, extending the approximation hardness of set cover by combining multiprover protocols with zero-knowledge techniques, we show that for every $\epsilon > 0$, a $(1 - \epsilon)\ln n$-approximation implies that $NP \subseteq DTIME(n^{O(\log \log n)})$. This makes domatic number the first natural maximization problem (known to the authors) that is provably approximable to within polylogarithmic factors but no better.

We also show that every graph has a domatic partition with $(1 - o(1))(\delta + 1)/\ln \Delta$ dominating sets, where the "$o(1)$" term goes to zero as Δ increases. This can be turned into an efficient algorithm that produces a domatic partition of $\Omega(\delta/\ln \Delta)$ sets.

Key words. domatic number, domination, approximation algorithms, probabilistic analysis

AMS subject classifications. 0569, 0585, 68Q17, 68W20, 68W40

PII. S0097539700380754

1. Introduction.

A *dominating set* in a graph is a set of vertices such that every vertex in the graph either is in the set or has a neighbor in the set. A *domatic partition* is a partition of the vertices so that each part is a dominating set of the graph. The *domatic number* of a graph is the maximum number of dominating sets in a domatic partition of the graph or, equivalently, the maximum number of disjoint dominating sets.

The domatic partition problem is one of the classical NP-hard problems. It is also one of the few graph problems in Garey and Johnson [14] whose approximability status on general graphs has until now been a blank page, with no published upper or lower bounds found in a literature search. The purpose of this paper is to mend that situation and derive the optimal approximability within a lower order term.

The domatic partition problem arises in various situations of locating facilities in a network. Assume that a node in a network can access only resources located at

*Received by the editors November 13, 2000; accepted for publication (in revised form) March 13, 2002; published electronically December 11, 2002. This work appears in preliminary form in *Proceedings of the* 32*nd Annual ACM Symposium on Theory of Computing*, 2000, pp. 134–143 and *Proceedings of the* 12*th ACM-SIAM Symposium on Discrete Algorithms*, 2001, pp. 922–923.

http://www.siam.org/journals/sicomp/32-1/38075.html

†Department of Computer Science and Applied Mathematics, The Weizmann Institute, Rehovot, Israel (feige@wisdom.weizmann.ac.il). This author's research was supported in part by a Minerva grant.

‡Department of Computer Science, University of Iceland, IS-107 Reykjavík, Iceland (mmh@hi.is). Part of this author's work was done while visiting the School of Informatics, Kyoto University, Japan.

§Department of Computer Science, Rutgers University, Camden, NJ (guyk@crab.rutgers.edu).

¶Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 (srin@cs.umd.edu). Part of this author's work was done while at Bell Laboratories, Lucent Technologies, 600–700 Mountain Avenue, Murray Hill, NJ 07974-0636. This author's research was supported in part by NSF award CCR-0208005.

neighboring nodes (or at itself). Then if there is an essential type of resource that must be accessible from every node (a hospital, a printer, a file, etc.), copies of the resource need to be distributed over a dominating set of the network. If there are several essential types of resources, each one of them occupies a dominating set. If each node has bounded capacity, there is a limit to the number of resources that can be supported. In particular, if each node can only serve a single resource, the maximum number of resources supportable equals the domatic number of the graph [13]. We can show how the general case of larger, possibly nonuniform, capacities can be reduced to the unit case.

We review some elementary facts about dominating sets and domatic partitions in light of the novelty of the problem to many readers. Dominating sets satisfy a monotonicity property with regards to vertex additions: if D is a dominating set and $D' \supset D$, then D' is also a dominating set. This implies that if a graph contains k disjoint dominating sets, then its domatic number is at most k; those nodes not belonging to any of the k sets can be arbitrarily added to the sets to form a proper partition of the vertex-set. The domatic number can then be alternatively defined as the maximum number of disjoint dominating sets. Every graph G satisfies $D(G) \geq 1$, and unless G contains an isolated node, $D(G) \geq 2$. On the other hand, $D(G) \leq \delta + 1$, where δ is the minimum degree; the reason being that a node of minimum degree must have some neighbor (or itself) in each of the disjoint dominating sets.

Fujita [12] has studied several greedy algorithms and shown that their performance ratio is no better than $(\delta + 1)/2$ for values of δ up to $O(\sqrt{n})$. The only other lower bound on $D(G)$ given in a recent encyclopedic treatment of domination problems [17, 16] is $D(G) \geq \lceil n/(n - \delta(G)) \rceil$ [37], where n is the number of vertices. This lower bound is relevant only in very dense graphs, since it degenerates to $D(G) \geq 2$ when $\delta(G) \leq n/2$.

A number of results are known for special classes of graphs. A graph G is said to be *domatically full* if $D(G) = \delta(G) + 1$, the maximum possible. Determining if a d-regular graph is domatically full is NP-complete, for any $d \geq 3$ [32, 21]. Farber [8] showed nonconstructively that *strongly chordal* graphs are domatically full. This class contains the classes of interval graphs and path graphs. Rao and Rangan [35] then gave a linear-time algorithm for interval graphs, and Peng and Chang [30] for strongly chordal graphs. Farber's theorem turned out to be a special case of a result of Berge [5] for *balanced* hypergraphs, and Kaplan and Shamir [19] presented a simple algorithm. They also showed split graphs and bipartite graphs to be NP-hard. Efficient algorithms are known for partial k-trees, using generic methods [2]. Bonucelli [6] showed that circular-arc graphs are NP-hard, while Marathe, Hunt, and Ravi [26] gave a 4-approximation algorithm.

Let Δ denote the maximum degree of a given graph. Our main result is a tight bound on the approximability of the domatic number problem in general graphs. In particular, we give

(A) an algorithm that finds a domatic partition of size $(1 - o(1))(\delta + 1)/\ln n$, where the "$o(1)$" term goes to zero as n increases;
(B) an algorithm that finds a domatic partition of size at least $\delta/c \ln \Delta$, for some constant c;
(C) a nonconstructive argument showing that the domatic number is at least $(1 - o(1))(\delta + 1)/\ln \Delta$, where the "$o(1)$" term goes to zero as Δ increases. This shows that the *value* of the domatic number can be approximated within a factor of nearly $\ln \Delta$;

(D) a bound on the domatic number of random graphs, showing that for most graphs, the domatic number is at most $(1 + o(1))(\delta + 1)/\ln \Delta$, where the "$o(1)$" term goes to zero as n increases;

(E) a construction showing that for every $\epsilon > 0$, no polynomial-time algorithm can approximate the domatic number problem within a $(1 - \epsilon) \ln n$ factor, unless NP has slightly superpolynomial-time algorithms ($NP \subseteq DTIME(n^{\log \log n})$). It also yields a $(1 - o(1)) \ln \Delta$-hardness. These results hold even for bipartite graphs and split graphs.

The $(1 + o(1)) \ln n$-approximation algorithm is a simple randomized assignment (though care is needed not to lose a factor of two in the analysis) and is derandomized using the method of conditional probabilities. The results (B) and (C) above use the Lovász local lemma (LLL) [7] as their basic tool. Suitable application of the LLL to our randomized assignment algorithm above shows that the domatic number is at least $(1/3 - o(1))(\delta + 1)/\ln \Delta$; we then refine this using a "slow partitioning" scheme, leading to our result that the domatic number is at least $(1 - o(1))(\delta + 1)/\ln \Delta$. The $O(\ln \Delta)$-approximation algorithm is a constructive version of the LLL, following an approach of Beck [4].

The hardness construction builds on the proof of Feige [9] of similar hardness for the set cover and dominating set problems. In fact, the construction here generalizes the result of [9] in that it shows that it is hard to distinguish between the following two cases: when the minimum dominating set is large (and thus the domatic number small) or when there are many small disjoint dominating sets. This parallels the situation with the archetypical *minimum* partitioning problem, graph coloring, where Feige and Kilian [11] showed that it is hard to distinguish between the case when the maximum independent set is small and when the chromatic number is small. The construction of the current paper, in fact, draws additionally on the zero-knowledge techniques used in [11].

It is instructive to view our results in a larger context—that of the study of approximation algorithms in general. It has been empirically observed and further supported by classification of constraint satisfaction problems [20] that there seem to be no "natural" maximization problems approximable within polylogarithmic factors but no better. Our results provide (to the best of our knowledge) the first maximization problem with such a behavior, as the domatic number is a maximization problem approximable within logarithmic factors but no better.

Our algorithmic results give absolute ratios, namely bounds in terms of some basic parameters of the graph (minimum degree, number of vertices) rather than in terms of the size of the optimal solution. These are, in fact, the first nontrivial lower bounds on the size of an optimal domatic partition for arbitrary δ, Δ such that $\delta \geq \ln \Delta$:

$$D(G) \geq (1 - o(1)) \cdot \frac{\delta + 1}{\ln \Delta}. \tag{1}$$

As shown in section 2.5, this bound is best possible up to lower order terms, for a large range of values of $\delta = \delta(n)$ and $\Delta = \Delta(n)$.

In the past, most absolute ratios have been obtained by fairly simple greedy algorithms. Our algorithms are derandomizations of simple randomized algorithms, but their derandomized versions are not particularly natural, and natural greedy algorithms for the problem attain much worse results. It is also interesting that the hardness result gives a "gap location at 1": namely, it is equally hard to approximately partition graphs that are domatically full.

The rest of the paper is divided into positive results—algorithmic and existential—on domatic partitions in section 2 and hardness results in section 3.

2. Approximation algorithms and existential results. This section is devoted to positive results for domatic partitions. In section 2.1, we give an algorithmic proof of the bound $D(G) \geq (1 - o(1))(\delta + 1)/\ln n$ and also show that $D(G) \geq (1/3 - o(1))(\delta + 1)/\ln \Delta$. (The two usages of "$o(1)$" here, respectively, correspond to $n \to \infty$ and $\Delta \to \infty$.) This second result is made algorithmic in section 2.2, with a loss in the constant factor. The existential result that $D(G) \geq (1 - o(1))(\delta + 1)/\ln \Delta$ is then shown in section 2.3 and shown to be tight on random graphs in section 2.5. Section 2.4 is devoted to a short analysis of the natural greedy algorithm for domatic partition.

Notation. Let $N(v)$ denote the set of neighbors of a vertex v in the given graph G, and let $N^+(v) = \{v\} \cup N(v)$. Let $d(v) = |N(v)|$ denote the degree of v, and let $d^+(v) = |N^+(v)| = 1 + d(v)$. A *partial coloring* of G is an arbitrary coloring of an arbitrary subset of the vertices. Given a current partial coloring, define a Boolean variable $A_{v,c}$ to be true if there is no vertex of color c in $N^+(v)$ and to be false otherwise. Note that the events $A_{v,c}$ are "bad events" for us: if $A_{v,c}$ holds for some pair (v, c), then the coloring is not a domatic partition; conversely, if none of the events $A_{v,c}$ hold, then every vertex v "sees" every color in $N^+(v)$, and we will have a domatic partition. Thus, our focus will be on avoiding all of these bad events.

Define $[\ell]$ to be the set $\{1, 2, \ldots, \ell\}$. For an event X, $\mathcal{P}[X]$ denotes its probability and $\mathbf{E}[X]$ its expectation. Finally, let e denote the base of the natural logarithm.

2.1. Logarithmic bounds.

THEOREM 1. *Any graph admits a (polynomial-time constructible) domatic partition of size* $(\delta + 1)(1 - O(\log\log n/\log n))/\ln n$.

Proof. Independently give each vertex one of $\ell = (\delta + 1)/\ln(n \ln n)$ colors at random. For any vertex-color pair (v, c), $\mathcal{P}[A_{v,c}] = (1 - 1/\ell)^{d^+(v)} \leq e^{-d^+(v)/\ell} \leq 1/(n \ln n)$. Thus, summing over all (v, c) pairs, the expected total number of bad events $A_{v,c}$ is at most $\ell/\ln n$. Hence, the expected number of colors that form dominating sets is at least

$$\ell - \frac{\ell}{\ln n} = \frac{\delta + 1}{\ln n}\left(1 - \frac{\ln\ln n + 1}{\ln(n \ln n)}\right). \tag{2}$$

The color-classes that do not form dominating sets can all be merged into any one color-class that is a dominating set; thus, we get a domatic partition whose expected number of sets is at least as large as the right-hand side of (2).

This randomized argument can be derandomized using the method of conditional probabilities (cf. [1]). Arbitrarily number the vertices as $v_1, v_2, \ldots, v_n$, and color the vertices in this order (never recoloring a vertex) as follows. Color v_1 arbitrarily. Suppose the first $j \geq 1$ vertices have been colored with respective colors $c_1, c_2, \ldots, c_j$; vertex v_{j+1} is colored as follows. Let $d_{j+1}(v) = |N^+(v) \cap \{v_{j+1}, v_{j+2}, \ldots, v_n\}|$. Then, the conditional probability of the bad event $A_{v,c}$ is given by

$$\mathcal{P}[A_{v,c} | c_1, c_2, \ldots, c_j] = \begin{cases} 0 & \text{if } \exists v_z \leq j \text{ such that } v_z \in N^+(v) \text{ and } c_z = c, \\ (1 - 1/\ell)^{d_{j+1}(v)} & \text{otherwise.} \end{cases}$$

The weight of the current coloring is given by

$$g(c_1, c_2, \ldots, c_j) \doteq \sum_{v \in V} \sum_{c} \mathcal{P}[A_{v,c} | c_1, c_2, \ldots, c_j];$$

this is precisely the expected number of (v, c) pairs for which $A_{v,c}$ will hold after coloring all vertices, given the current coloring $c_1, \ldots, c_j$. In each step $j+1$, we choose a color for v_{j+1} so that the weight of the coloring does not increase. Such a color exists, since $g(c_1, c_2, \ldots, c_j)$ is a convex combination of the values $\{g(c_1, c_2, \ldots, c_j, c_{j+1}) : c_{j+1} \in [\ell]\}$:

$$g(c_1, c_2, \ldots, c_j) = (1/\ell) \cdot \sum_{c_{j+1} \in [\ell]} g(c_1, c_2, \ldots, c_j, c_{j+1}).$$

Then, the total number of colors that are not dominating sets is at most the weight of the final coloring, which we ensure is at most the expected number at the outset, or $\ell / \ln n$. □

We now refine this argument using the LLL to get better bounds when $\Delta \leq n^{1/3}$. We state the symmetric, simpler version of the LLL.

LEMMA 2 (LLL [7]). *Let $p < 1$, and let $\mathcal{E}_i$, $1 \leq i \leq k$, be k events such that $\mathcal{P}[\mathcal{E}_i] \leq p$ for all i. Suppose there is an integer d such that $e \cdot p \cdot (d+1) \leq 1$, and each event is independent of all but at most d other events. (More precisely, for each $\mathcal{E}_i$, there is a set T_i of at least $k - d - 1$ other events $\mathcal{E}_j$, such that the conditional probability of $\mathcal{E}_i$ given any Boolean combination of the events in T_i equals the unconditional probability of $\mathcal{E}_i$.) Then, $\mathcal{P}[\bigwedge_i \bar{\mathcal{E}}_i] > 0$.*

Independently color each vertex randomly with one of $\ell = \lfloor (\delta+1)/(3 \cdot \ln(3^{1/3} \cdot \Delta)) \rfloor$ colors. For each (v, c) pair, $\mathcal{P}[A_{v,c}] \leq (1 - 1/\ell)^{d^+(v)} \leq 1/(3 \cdot \Delta^3)$. We note that each event $A_{v,c}$ is independent of all but at most $\ell(1 + d(v) + d(v) \cdot (\Delta - 1))$ other such events, since vertices of distance at least 3 from v are completely irrelevant for v. More precisely, in the notation of Lemma 2, we can take $T_{v,c}$ to be the set of all events of the form (w, c'), where w is a vertex at a distance of at least 3 from v, and where c' is any color from $[\ell]$: conditioning on any Boolean combination of the events in $T_{v,c}$ does not influence the colors chosen by vertices in $N^+(v)$. The following lemma now directly follows from the LLL, using the fact that $\ell < \Delta$ and $d(v) \leq \Delta$: we set $d = \Delta^3 - 1$ and $p = 1/(3 \cdot \Delta^3)$ in using the LLL.

LEMMA 3. *Any graph admits a domatic partition of size $(1/3 - o(1))\delta / \ln \Delta$, where the "$o(1)$" term tends to zero as $\Delta \to \infty$.*

In section 2.3, we will refine the above approach by conducting a two-stage partitioning that attains the tight value of $1 - o(1)$ instead of the value $1/3 - o(1)$ of Lemma 3. However, the above direct approach will help us develop a simple algorithmic version of Lemma 3 in section 2.2. It also motivates the reason for developing the approach of section 2.3.

Remark. It is interesting to note that the Δ in the bound of Lemma 3 cannot be replaced by δ nor by $\overline{d}$, the average degree. Consider, for example, the bipartite graph with $3 \cdot \delta$ vertices on the left side and $\binom{3 \cdot \delta}{\delta}$ vertices on the right side, with each vertex on the right side connected to a particular subset of δ vertices in the left side. The domatic number of this graph is two. Indeed, say that there are 3 disjoint dominating sets. One of these sets, S, contains at least δ-vertices on the left side. There exists a vertex v on the right side all of whose neighbors are in S. Hence, the two remaining sets must both contain the vertex v, a contradiction.

A parameter that is intermediate between average and maximum degree is the *inductiveness* $\Delta^*(G) = \max_{H \subseteq G} \delta(H)$. It is most notable for giving a tighter upper bound on the chromatic number than the maximum degree, as $\chi(G) \leq \Delta^*(G) + 1 \leq \Delta(G) + 1$. The above construction shows, however, that Δ^* cannot replace Δ in the bound of Lemma 3 on the domatic number.

2.2. $O(\log \Delta)$-approximation algorithm. We use an algorithmic version of the LLL due to Beck [4] that derandomizes the probabilistic argument with some loss in the constants. We may assume without loss of generality that G is connected, since we can treat the connected components separately. Our algorithm has three phases assigning colors to successively larger fractions of the vertices. After the first phase, each vertex is either fully satisfied, seeing all colors within its neighborhood, or has at most one third of its neighbors colored. We show that the subgraph induced by nodes that are still active, i.e., either unsatisfied or not yet colored, consists with high probability of only small connected components of $O(\Delta^6 \log n)$ vertices each. After the second phase, more vertices are colored, with at most two thirds of the neighbors of yet unsatisfied vertices being colored. The connected components induced by active vertices are now of only $O(\Delta^7 \log\log n)$ size. Then, depending on the value of Δ, we can either solve each component by exhaustive search or apply Theorem 1 to obtain a full coloring where each vertex is satisfied.

2.2.1. The algorithm. The first phase proceeds as follows. Given a coloring of some of the vertices, call a vertex v *dangerous* iff

1. at least $\delta/3$ neighbors of v have been colored, and
2. not all the ℓ colors appear in the neighborhood of v.

Let $\ell = \delta/(c \ln \Delta)$ for a suitably large constant c. Order the vertices arbitrarily as $v_1, v_2, \ldots, v_n$ and process them in this order. When processing v_i, we do the following. If v_i or one of its neighbors is dangerous now, we *freeze* v_i; otherwise we independently assign it one of ℓ colors at random.

When the process ends, some vertices are colored and some are frozen, and some are dangerous and some are not. The vertices that are not dangerous belong to one of two categories:

Good: A good vertex sees all colors in its neighborhood.

Neutral: A neutral vertex v does not see all colors in its neighborhood but is not dangerous. This can happen only if more than $2/3$ of v's neighbors were frozen.

Thus, we have two orthogonal partitions: colored/frozen and good/neutral/dangerous.

Vertices that are both good and colored do not need to be considered further in the later phases. Call the other vertices *saved*, i.e., those that are dangerous, frozen, or neutral. As in [4], we are interested in the maximum size of a connected component of the subgraph induced by saved vertices, as this bounds the size of the independent subproblems in the next phase. We show in section 2.2.2 that with probability at least $1/2$, the largest connected component in the saved graph has size $O(\Delta^6 \log n)$; let us assume that this size bound holds.

Phase two of the algorithm is run separately on each connected component induced by the saved vertices. Note that each dangerous or neutral vertex v has at least $d(v) - \delta/3$ frozen (i.e., uncolored) neighbors in its connected component. Phase two differs from phase one in that some of the vertices are colored before we begin. We leave these colors untouched, because they may be useful for the vertices that were not saved. We only color the frozen vertices, and again define the notion of phase two dangerous, frozen, and neutral vertices. Since the number of vertices we are dealing with now in any connected component is $O(\Delta^6 \log n)$, we have essentially replaced n by $O(\Delta^6 \log n)$. Thus, analysis similar to that of phase one shows that the connected components of the newly saved vertices has size at most $N = O(\Delta^6(\log \Delta + \log\log n))$ with probability at least $1/2$.

In phase three, each dangerous/neutral vertex has at least $d(v)-2\delta/3 \geq \delta/3$ frozen neighbors in its connected component. If $\Delta > \log\log n$, we have a domatic partition of the frozen vertices to roughly $\delta/(3\ln N)$ parts via Theorem 1. As $\ln N = O(\log \Delta)$, this is good enough. If $\Delta \leq \log\log n$, then we can find a domatic partition of size $\Omega(\delta/\ln\Delta)$ whose existence is guaranteed by Theorem 3, using an exhaustive search. Since $\Delta \leq \log\log n$, this only takes time

$$N^{O(\delta/\ln\Delta)} \leq (\log\log n)^{O((\log\log n))} \leq poly(n).$$

This completes the description of the algorithm.

2.2.2. Analysis of the algorithm. We now show that with probability at least 1/2, the largest connected component in the saved graph in the first phase has size $O(\Delta^6 \log n)$. This will yield a proof of correctness of our algorithm.

Let $X(u)$ be the indicator random variable for vertex u becoming dangerous, and let $q = \ell(1-1/\ell)^{\delta/3}$.

LEMMA 4. *Let $U = \{u_1, u_2, \ldots, u_k\}$ be any set of vertices with pairwise distance at least 3. Then,* $\Pr[X(u_1) = X(u_2) = \cdots = X(u_k) = 1] \leq q^k$.

Proof. If a vertex v is a neighbor of the set U, then it has a unique neighbor in U since the elements of U have pairwise distance at least 3. Let $\vec{a} = (a_1, a_2, \ldots, a_k)$ be any sequence of k colors, and let S_i be the random variable denoting the set of the first i nonfrozen neighbors of U. Let $D_i(\vec{a})$ be the event that "for all $j = 1, 2, \ldots, k$, all neighbors of u_j in S_i avoid color a_j." Thus, $D_i(\vec{a})$ is the event that even after processing the first i nonfrozen neighbors of U, each u_j in U was missing a particular color a_j. We may assume without loss of generality that sets S_i exist for every i up to $\delta k/3$, because otherwise some vertex u in U does not have $\delta/3$ nonfrozen neighbors, and then u cannot be dangerous. Hence, $D_i(\vec{a})$ is defined for all $i \leq \delta k/3$. We have $\Pr[D_0(\vec{a})] = 1$. Now for every $i < \delta k/3$, $\Pr[D_{i+1}(\vec{a})] = \Pr[D_i(\vec{a})] \cdot (1-1/\ell)$, because the color of the $(i+1)$st nonfrozen neighbor of U is chosen at random independent of the previous colors and independent of which vertex it happens to be. Hence, $\Pr[D_i(\vec{a})] = (1-1/\ell)^i$; so, $\Pr[\exists \vec{a} :\ D_{\delta k/3}(\vec{a})] \leq \ell^k (1-1/\ell)^{\delta k/3} = q^k$.

If all the u_j are dangerous, then there is some $\vec{a}$ for which $D_{\delta k/3}(\vec{a})$ is true; this completes the proof. ∎

To prove that the largest connected component in the saved graph is "small enough" with reasonable probability, we now show that with reasonable probability the maximum number of vertices in a spanning tree of such a component is "small." This is done as follows. By a standard argument, a large connected component contains many vertices with a particular minimum pairwise distance. We first prove that the number of vertices with large pairwise mutual distance which are all saved is "small." This indirectly bounds the maximum number of vertices in a connected component as a function of Δ, which is enough for our purposes.

A set of vertices is said to be 7-*separated* if it is of mutual distance at least 7. A 7-separated set of k vertices is said to be a *bad k-set* if, additionally, it becomes connected if we connect all vertices of distance exactly 7. As shown next, the number of such sets in G is at most

$$n(4\Delta^7)^k. \tag{3}$$

Consider a spanning tree on the set where vertices of distance exactly 7 are connected. The number of distinct shapes of trees on k vertices is at most 4^{k-1}. Namely, such a tree can be uniquely represented by ordering the vertices in a lexicographic breadth-first order and attaching two flag bits to each nonroot vertex, whether it has the same

parent as the previous vertex in the order, and whether it has a child or not. For each shape of a tree, there are n possibilities of choosing the root and thereafter at most Δ^7 possibilities of choosing each new vertex since we already chose its parent in the tree.

Let $Y(u)$ be the indicator random variable for vertex u becoming saved. To complete our argument that no connected component of the saved vertices is "large," we show the following lemma.

LEMMA 5. *For any 7-separated set of vertices $v_1, v_2, \ldots, v_k$,*

$$\Pr[Y(v_1) = Y(v_2) = \cdots = Y(v_k) = 1] \leq (2.5\Delta q)^k,$$

where q is as defined prior to Lemma 4.

Proof. The following definition will be useful for this proof:

$$Z(u) \doteq X(u) + \left(\sum_{v \in N(u)} X(v) \right) + \frac{\sum_{v \in N(u)} \sum_{w \in N(v)} X(w)}{2d(u)/3}.$$

If vertex u is saved, then we have one of three cases: (i) u is dangerous and thus $X(u) = 1$; (ii) u has a dangerous neighbor and so $\sum_{v \in N(u)} X(v) \geq 1$; or (iii) u is neutral, so at least $2d(u)/3$ of its neighbors are frozen, and by the preceding argument it holds for each frozen neighbor v of u that $\sum_{w \in N(v)} X(w) \geq 1$. Therefore, we have the simple but useful observation that *if $Y(u) = 1$, then $Z(u) \geq 1$.*

By Markov's inequality,

$$\Pr[Y(v_1) = Y(v_2) = \cdots = Y(v_k) = 1] \leq \Pr\left[\prod_{i=1}^{k} Z(v_i) \geq 1 \right] \leq \mathbf{E}\left[\prod_{i=1}^{k} Z(v_i) \right]. \tag{4}$$

Now, because $Z(\cdot)$ is linear in the $X(\cdot)$ and using the linearity of expectation, we can expand $\mathbf{E}[\prod_i Z(v_i)]$ as a linear combination of the terms $\Pr[X(w_1) = X(w_2) = \cdots = X(w_k) = 1]$. The main observation is that since the v_i have pairwise distance at least 7, the w_i have pairwise distance at least 3. Thus, by Lemma 4, any such term has probability at most q^k. Thus we get

$$\mathbf{E}\left[\prod_i Z(v_i) \right] \leq \left(q + \Delta q + \frac{d(u)(\Delta - 1)q}{2d(u)/3} \right)^k \leq (2.5\Delta q)^k$$

by first replacing the probability of intersection of events by the product of probabilities and then unfolding and reversing the above expansion. ∎

Consider now a connected component in the subgraph of G of saved vertices. If its size is $k\Delta^6$ or more, then it must contain a bad k-set (which is obtained by repeating the procedure of putting a vertex in the bad k-set and removing all vertices of distance at most 6). Setting the constant c in the expression $\ell = \delta/(c \ln \Delta)$ large enough, we get that $q \leq \Delta^{-9}/10$. Set $k = \log(2n)/\log \Delta$, and recall (3) and Lemma 5. We get that the probability of existence of a connected component of the saved vertices with cardinality at least $k\Delta^6$ is at most $(2.5q\Delta)^k n(4\Delta^7)^k \leq 1/2$.

Finally, the above Las Vegas algorithm can be derandomized by the approach of *pessimistic estimators* [33], which is a generalization of the method of conditional probabilities. Briefly, we proceed as follows. As before, we process the vertices one-by-one. Suppose it is currently the turn of vertex v. If v is frozen, we skip over it.

Otherwise, we deterministically choose a color for it that minimizes the probability of emergence of a large connected component of the saved vertices using our bounds derived above. Since $k = \log(2n)/\log\Delta$, we see from (3) that the number of possible bad k-sets to be considered in our analysis above is bounded by a polynomial in n. Hence, we can write down a pessimistic estimator and choose, in deterministic polynomial-time, a color for vertex v that minimizes the pessimistic estimator.

2.3. Getting the right constant. Our next result improves the value $1/3-o(1)$ of Lemma 3 to the existentially best possible value of $1 - o(1)$.

THEOREM 6. *There is a constant $a > 0$ such that for large enough Δ_0, for every graph G with $\Delta \geq \Delta_0$, $D(G) \geq \lfloor \frac{\delta}{\ln\Delta + a\ln\ln\Delta} \rfloor$.*

We remark that if $\Delta < \Delta_0$, the theorem holds by setting a large enough. For the special case of $\Delta \leq 2$, the value of $D(G)$ is well known via a simple case analysis; there is also a linear-time algorithm for the domatic partition problem if $\Delta \leq 2$.

Proof. For the rest of section 2.3, any "$o(1)$" term will denote a function of Δ alone that goes to zero as Δ increases. We prove the theorem for a being any constant greater than 7, for all large enough Δ; this choice of a can be further improved, but we do not attempt this optimization here.

Preprocessing. We preprocess the graph as follows. As long as there is an edge that has both end-points with degree more than δ, remove such an edge from the graph. At the end of this process, the minimum degree remains at δ, and the maximum degree is at most Δ. We will now show a lower bound on the domatic number of this preprocessed version, which clearly will yield the same lower bound on the domatic number of the given graph. (This is because the preprocessing only removes edges from the given graph.) The useful property that now holds is that for each vertex u, either $d(u) = \delta$, or for all neighbors v of u, $d(v) = \delta$.

There are two cases, the first one being simpler.

Case I: $\delta \leq \ln^4\Delta$. We assume that $\delta > \ln\Delta + a\ln\ln\Delta$, since the theorem is trivially true otherwise. Define

$$\ell = \left\lfloor \frac{\delta}{\ln\Delta + a\ln\ln\Delta} \right\rfloor.$$

Color each vertex with a random color from $[\ell]$, independent of all other vertices. We will now use the LLL to show that $\mathcal{P}[\bigwedge_{u,c} \bar{A}_{u,c}] > 0$ in the same way as we did for Lemma 3. For each (u, c), we have

$$\Pr[A_{u,c}] = \left(1 - \frac{1}{\ell}\right)^{d^+(u)} \leq e^{-\ln(\Delta(\ln\Delta)^a)\cdot d^+(u)/\delta} \leq (\Delta(\ln\Delta)^a)^{-1}. \tag{5}$$

Let $N_2(u)$ denote the set of vertices at a distance of 0, 1, or 2 from u in G. As in our proof of Lemma 3, each event $A_{u,c}$ depends only on events $A_{v,c'}$ with $v \in N_2(u)$; so, it depends on at most $|N_2(u)| \cdot \ell$ other such events. Our preprocessing step above helps bound $|N_2(u)|$ for all u. If $d(u) = \delta$, then $|N_2(u)| \leq 1+\delta+\delta(\Delta-1)$. If $d(u) > \delta$, our preprocessing ensures that $d(v) = \delta$ for all neighbors v of u; so, $|N_2(u)| \leq 1 + \Delta + \Delta(\delta - 1)$. Thus, $|N_2(u)| \leq \delta\Delta + 1 \leq \Delta\ln^4\Delta + 1$ for all u, since $\delta \leq \ln^4\Delta$. So, each $A_{u,c}$ depends on at most $O(\ell\Delta\ln^4\Delta) = O(\Delta\ln^7\Delta)$ other such events. Recalling the LLL and (5), we see that $\mathcal{P}[\bigwedge_{u,c} \bar{A}_{u,c}] > 0$ as required, since $a > 7$.

Case II: $\delta > \ln^4\Delta$. Let $\epsilon = 1/(\ln\Delta)$; define $\ell_1 = \lfloor \epsilon^3\delta \rfloor$ and $\ell_2 = \lfloor \ln^2\Delta/(1 + b(\ln\ln\Delta)/\ln\Delta) \rfloor$, where b is any constant larger than 5. We will show the existence

of a domatic partition of size $\ell_1\ell_2$, i.e., a coloring of V using $\ell_1\ell_2$ colors, in such a way that for every vertex u there is at least one vertex of each color in $N^+(u)$. (Recall that $\delta > \ln^4 \Delta$. By choosing $b < 6$, for instance, we can ensure that $\ell_1\ell_2 \geq \delta/(\ln\Delta + 6\ln\ln\Delta)$ for all large enough Δ.) It will be convenient to view the colors as elements of $[\ell_1]\times[\ell_2]$. We will apply a two-stage coloring: the first coloring determines the first components of the vertex-colors, and the second coloring is for the second components. We can view the first coloring as a coarse partition, which the second coloring turns into a fine partition. The primary purpose of the first coloring is to reduce the dependencies sufficiently for our analysis of the second coloring.

The first partitioning is as follows. Color each vertex with a random color from $[\ell_1]$, independent of all other vertices. For each vertex u and each color c, define $X^+_{u,c}$ to be the subset of $N^+(u)$ that receives color c. We have $\mathbf{E}[|X^+_{u,c}|] = d^+(u)/\ell_1$. Let $B_{u,c}$ be the "bad" event that $\mid |X^+_{u,c}| - d^+(u)/\ell_1 \mid \geq 3\epsilon d^+(u)/\ell_1$. A Chernoff bound shows that

$$\Pr[B_{u,c}] \leq 2\cdot\exp(-(9/2-o(1))\cdot d^+(u)\epsilon^2/\ell_1) \leq \exp(-(9/2-o(1))\ln\Delta). \tag{6}$$

Once again, $B_{u,c}$ is independent of any Boolean combination of events of the form $B_{v,c'}$ for vertices v at a distance of 3 or more from u. Thus, each $B_{u,c}$ "depends" on $o(\Delta^3)$ other such events. Recalling (6), the LLL shows that $\Pr[\bigwedge_{u,c}\overline{B_{u,c}}] > 0$.

Fix a coloring $\chi_1 : V \to [\ell_1]$ which avoids all the events $B_{u,c}$. Choose a random color $\chi_2(u) \in [\ell_2]$ for each u, independent of all other vertices; the final color of u is the pair $(\chi_1(u), \chi_2(u))$. Let $\mathcal{B}_{u,c_1,c_2}$ be the bad event that there is no vertex of color (c_1, c_2) in $N^+(u)$. We now use the LLL to show that all these bad events can be avoided with positive probability.

For each vertex u and each $c \in [\ell_1]$, let $N^+_{u,c} = \{v \in N^+(u) :\ \chi_1(u) = c\}$, and define $d^+_{u,c} = |N^+_{u,c}|$. Since χ_1 avoids all the events $B_{u,c}$, we have

$$\forall (u,c),\ (1-3\epsilon)d^+(u)/\ell_1 \leq d^+_{u,c} \leq (1+3\epsilon)d^+(u)/\ell_1. \tag{7}$$

Fix an event $\mathcal{B}_{u,c_1,c_2}$. First,

$$\begin{aligned}\Pr[\mathcal{B}_{u,c_1,c_2}] = \left(1-\frac{1}{\ell_2}\right)^{d^+_{u,c_1}} &\leq e^{-(1-3\epsilon)d^+(u)/(\ell_1\ell_2)} \\ &\leq (\Delta(\ln\Delta)^b)^{-(1-3\epsilon)} \leq O((\Delta(\ln\Delta)^b)^{-1});\end{aligned} \tag{8}$$

the first inequality here follows from (7). Next, which other events does $\mathcal{B}_{u,c_1,c_2}$ depend on? Given $S \subseteq V$, let $N^+(S) \doteq \bigcup_{v\in S} N^+(v)$. Note that $\mathcal{B}_{u,c_1,c_2}$ simply says that all elements of N^+_{u,c_1} got a $\chi_2(\cdot)$ value different from c_2. Thus, we can check that $\mathcal{B}_{u,c_1,c_2}$ depends only on the events in

$$S(u,c_1,c_2) = \{\mathcal{B}_{v,c'_1,c'_2} :\ v \in N^+(N^+_{u,c_1}) \text{ and } c'_1 = c_1\}. \tag{9}$$

More precisely, we claim that $\mathcal{B}_{u,c_1,c_2}$ is independent of any Boolean function of the events lying outside $S(u,c_1,c_2)$; this can be verified by seeing that

$$N^+_{u,c_1} \cap \left(\bigcup_{(v,c'_1,c'_2):\ \mathcal{B}_{v,c'_1,c'_2}\notin S(u,c_1,c_2)} N^+_{v,c'_1}\right) = \emptyset.$$

We now bound $|S(u,c_1,c_2)|$ in order to apply the LLL; once again, our preprocessing step will be of help. If $d(u) = \delta$, then $|N^+(N^+_{u,c_1})| \leq \Delta|N^+_{u,c_1}|$; this is at most

$O(\delta\Delta/\ell_1)$, by (7). If $d(u) > \delta$, our preprocessing ensures that $|N^+(N^+_{u,c_1})| \leq \delta|N^+_{u,c_1}|$; so, from (7), we again get that $|N^+(N^+_{u,c_1})| \leq O(\delta\Delta/\ell_1)$. Thus, $|S(u,c_1,c_2)| \leq O(\delta\Delta\ell_2/\ell_1) = O(\Delta \ln^5 \Delta)$. We can now apply the LLL, using (8) and the facts that (i) $b > 5$ and (ii) each bad event $\mathcal{B}_{u,c_1,c_2}$ depends on at most $|S(u,c_1,c_2)|$ others. This completes the proof. □

To see why our two-stage coloring helps, note that the "dependence" $|S(u,c_1,c_2)|$ in the second coloring above is only $\Delta^{1+o(1)}$, as compared to the dependence of $\Delta^{3+o(1)}$ that we could get in the direct coloring approach underlying Lemma 3. The constraint "$v \in N^+(N^+_{u,c_1})$" in (9) saves us a factor of $\Delta^{1-o(1)}$, and the constraint "$c'_1 = c_1$" saves another factor of $\Delta^{1-o(1)}$. That the first-stage coloring eliminates many dependencies in this fashion is the main idea motivating this approach.

It is an open question if a domatic partition of the size guaranteed by Theorem 6 can be found in polynomial-time.

2.4. Greedy algorithm. One natural approach to the domatic partition problem is to try to greedily choose small dominating sets. The greedy algorithm iteratively pulls out dominating sets from the graph until the remainder is no longer dominating. The dominating sets are found by a standard $O(\log n)$-approximate greedy algorithm [18, 22].

LEMMA 7. *Suppose $D(G) = n/k$. Then, the greedy algorithm finds a domatic partition of $\Omega(n/(k^2 \log n))$ sets.*

Proof. We count how many disjoint dominating sets our algorithm finds before the set of vertices in them intersects at least half of the n/k vertex-disjoint dominating sets in the graph. During this period, there are at least $n/(2k)$ disjoint dominating sets; thus in each step there exists a dominating set of size at most $2k$, and we find one of size at most $2k \ln n$. Hence, in each step, a vertex from at most $2k \ln n$ different dominating sets is removed. It then requires at least $(n/2k)/(2k \ln n)$ steps to halve the original number of dominating sets in the graph. □

Since $D(G) \leq n$, the approximation ratio is maximized when $k \approx \sqrt{n/(4 \ln n)}$.

COROLLARY 8. *The performance ratio of the greedy domatic partition algorithm is $O(\sqrt{n \ln n})$.*

Fujita [12] has shown examples where the performance of this and some other greedy algorithms is $\Omega(\sqrt{n})$.

2.5. The domatic number of random graphs. We now show that the bound (1) is tight for a large range of values of $\delta = \delta(n)$, by studying $D(G)$ for suitable random graphs G. Suppose G is drawn from the random graph model $G(n,p)$; i.e., we take n labeled vertices, and put an edge with probability p independently between each pair of vertices. We will show that with probability $1 - o(1)$, $D(G) \leq (1 + o(1))\delta(G)/\ln \Delta(G)$. (Throughout this section, the "$o()$" and "$\omega()$" notation refers to n getting large.)

Choose any $p = p(n)$ such that $np = (\ln n)^{\omega(1)}$ and $p = o(1)$. It is easy to check via a Chernoff bound that with probability $1 - o(1)$, both δ and Δ will lie in the range $(1 \pm o(1))np$ for our random graph G. Fix any constant $\epsilon > 0$, and let $s = \lfloor (1-\epsilon)\ln(np)/p \rfloor$. We will show that with probability $1 - o(1)$, any dominating set in G will have size more than s. (Thus, with probability $1 - o(1)$, we will have $D(G) \leq n/(s+1)$, completing the proof.) For any given subset of the vertices S with $|S| = s$,

$$\Pr[S \text{ is a dominating set}] = (1 - (1-p)^s)^{n-s}$$

$$\begin{aligned}
&\leq (1 - e^{-(1+\Theta(p))sp})^{n-s} \\
&\leq (1 - (np)^{\epsilon-1-\Theta(p)})^{n-s} \\
&\leq e^{-(n-s)\cdot(np)^{\epsilon-1-\Theta(p)}} \\
&\leq e^{s-(1/p)\cdot(np)^{\epsilon-\Theta(p)}} \\
&= e^{s-(1/p)\cdot(np)^{\epsilon-o(1)}}.
\end{aligned}$$

Thus, the probability of existence of a dominating set of size s is at most

$$\begin{aligned}
\binom{n}{s} \cdot e^{s-(1/p)\cdot(np)^{\epsilon-o(1)}} \leq (n^s/s!) \cdot e^{s-(1/p)\cdot(np)^{\epsilon-o(1)}} &= (e^s/s!) \cdot e^{s\ln n-(1/p)\cdot(np)^{\epsilon-o(1)}} \\
&= o(1);
\end{aligned}$$

the bound $s \ln n = o((1/p) \cdot (np)^{\epsilon-o(1)})$ follows from the definition of s and from the fact that $np = (\ln n)^{\omega(1)}$.

3. Hardness of approximating the domatic number. We say that a problem is *hard to approximate within ratio* ρ if having a polynomial-time (randomized) ρ-approximation algorithm for it would violate some standard hardness assumption, such as P $\neq$ NP. The hardness assumption that we use in this paper is that NP does not have (randomized) algorithms that run in time $n^{O(\log\log n)}$; for brevity we shall just use the term *hard to approximate*.

We shall prove the following theorem.

THEOREM 9. *For every fixed $\epsilon > 0$, it is hard to approximate the domatic number within a ratio of $(1-\epsilon)\ln|V|$.*

For this purpose, it is helpful to work with a related but different problem.

DEFINITION 1. *A* one-sided dominating set *in a bipartite graph $G(V_1, V_2, E)$ is a set of vertices $U \subseteq V_1$ such that for every $v \in V_2$ there is some $u \in U$ with $(u, v) \in E$. Here it is assumed that the intended bipartition (V_1, V_2) is given explicitly as part of the input and that every vertex in V_2 has some neighbor in V_1. Observe that this problem merely is a reformulation of the well-known set cover problem.*

The one-sided domatic number *of a bipartite graph is the maximum number of mutually disjoint one-sided dominating sets that the graph contains.*

Observe that the dominating set and domatic number problems have a relation similar to the one of the coloring versus maximum independent problem. A coloring is a packing of independent sets while a domatic partition is a packing of dominating sets.

A related problem is the *set cover* problem, where a collection of subsets $\mathcal{S}$ of a base set U is given, and we are to find a minimum cardinality subcollection that contains all elements of U. The *set cover packing number* is then the maximum number of mutually disjoint set covers. It is well known that minimum dominating set, minimum one-sided dominating set, and minimum set cover are strongly related, and that $\ln n$ is the best approximation ratio for all of them within lower order terms (details in section 3.2). The one-sided domatic number problem can be shown to be equivalent to the set cover packing problem in terms of optimization. We are not aware of a similar relationship between one-sided domatic number and domatic number. However, we can give a reduction that yields the necessary result.

PROPOSITION 10. *Let $c > 1$ and consider bipartite graphs $G(V_1, V_2, E)$ with $|V_1|$ large enough (e.g., $|V_1| \geq 4^c$), $|V_2| > |V_1|^c$, and the following* promise*: for some $0 \leq \epsilon \leq 1 - 1/c$ and for r and q satisfying $rq > (1-\epsilon)|V_1|\ln|V_2|$ (r and q may depend on the size of G), either*

- *the size of the smallest one-sided dominating set in G is at least r, or*
- *the one-sided domatic number of G is at least q.*

Note that the two cases for the promise cannot both hold. If it is hard to distinguish which of the two cases holds, then it is hard to approximate the domatic number within a ratio of $(1 - 1/c - \epsilon) \ln |V|$.

The proof of Proposition 10 is given in section 3.2. The main result of the section is the following theorem, proved in section 3.5.

THEOREM 11. *For every $\epsilon > 0$ and every integer $c > 1$ the two cases of Proposition* 10 *cannot be distinguished in (random) polynomial-time unless NP has (randomized) algorithms that run in time $n^{O(\log \log n)}$.*

Theorem 9 now follows from Theorem 11 and Proposition 10.

Remark. Theorem 9 implies a $\ln \Delta$ hardness of approximation result, for $\Delta \simeq n^{\psi}$ for some $0 < \psi < 1$ which is close to 1. To obtain $\ln \Delta$ hardness of approximation results when Δ is much smaller compared to n, simply make many disjoint copies of the graph, increasing n without changing Δ or the domatic number.

3.1. Overview and intuition. Before presenting the proof of Theorem 11, let us provide some background on proving hardness of approximation results in general and how hardness of approximation results were proved for problems related to domatic number.

A convenient starting point for proving hardness of approximation results is the problem of Max 3SAT. The input to this problem is a 3CNF formula and the desired output is an assignment to the variables that satisfies as many clauses as possible. The well known PCP theorem of [3] implies (or in fact, is equivalent to) the following *gap*: for some $\epsilon > 0$ it is NP-hard to distinguish between 3CNF formulas that are satisfiable (which we call *yes* instances) and 3CNF formulas in which every assignment satisfies at most a $(1 - \epsilon)$-fraction of the clauses (which we call *no* instances). This hardness result can be extended to a restricted version of Max 3SAT in which the input 3CNF formula has the property that each variable appears in exactly 5 clauses. (The choice of 5 is arbitrary here. Any other constant greater than 5 would do as well.) We call this restricted version Max 3SAT-5.

As noted above, the set cover problem is strongly related to the dominating set problem, which in turn is related to the domatic number problem. Moreover, the one-sided domatic number problem is equivalent to the set cover packing problem. Hence our plan for proving Theorem 11 is to take known results regarding the hardness of approximation of set cover and modify their proof so that it shows hardness of approximation for the set cover packing problem as well (and hence also for one-sided domatic number). To see more explicitly what needs to be done, let us first review at a very high level the known result [9] that set cover (and one-sided dominating set) is hard to approximate within a factor of $(1 - \epsilon) \ln n$.

The proof in [9] reduces instances of Max 3SAT-5 to instances of one-sided dominating set. The reduction is slightly super polynomial (instances of size n are mapped to instances of size $n^{O(\log \log n)}$) and is a *gap reduction* in the following sense: *yes* instances give bipartite graphs that have small one-sided dominating sets and *no* instances give graphs all of whose one-sided dominating sets are much larger. To prove hardness of approximation for one-sided domatic number, the requirement for *no* instances does not change, as it already implies a small one-sided domatic number. However, we would like *yes* instances to give bipartite graphs that have not just one small one-sided dominating set but many disjoint small one-sided dominating sets, and hence a large one-sided domatic number.

To achieve this extra property, we invoke an idea used in [11] when proving hardness of approximation for the chromatic number problem. In our context, it suffices to change the starting point of the reduction from the problem Max 3SAT-5 to the problem Max-3-colorability-5. This is the problem of coloring a 5-regular graph with 3 colors so as to maximize the number of edges legally colored (see section 3.3). It also has a gap similar to that of Max 3SAT-5. The important property of Max 3-colorability-5 is that *yes* instances of it necessarily have many "disjoint" solutions. When these instances are reduced to instances of one-sided dominating set, the resulting bipartite graph has many disjoint small one-sided dominating sets. This implies a large one-sided domatic number, as required by Theorem 11.

Hence, to complete the proof of Theorem 11, we need to accomplish three things.

1. Introduce the problem of Max 3-colorability-5 and its properties. This is done in section 3.3.

2. Give the reduction from Max 3-colorability-5 to one-sided dominating set. This reduction closely follows the reduction of [9] from Max 3SAT-5 to set cover, except for one small extra step (vertices on one side of the bipartite graph are duplicated many times, an operation that was not performed in the reduction to set cover). This reduction is fairly complicated, but essentially all complications come from the reduction in [9].

3. Prove the properties of the reduction. This has two parts. One is to show that *yes* instances of Max 3-colorability-5 are reduced to bipartite graphs with high one-sided domatic number, which is done in Lemma 17. The other is to show that *no* instances are reduced to bipartite graphs with only large one-sided dominating sets. This part is not proved in this paper, because the proof in [9] that *no* instances of Max 3SAT-5 give instances with a large set cover, extends virtually without change to our adaptation of the reduction of [9].

Appendix B reviews some of the ingredients of the reduction used in [9] from Max 3SAT-5 to set cover and explains some modifications used in our context. Some of these ingredients are only used in the analysis of what happens on *no* instances, and hence do not come into any of the proofs in this paper. They are included only in the overview so as to give some indication of what led to the construction of the reduction. For more details, see [9].

3.2. Domination and one-sided domination. The three problems, minimum dominating set, one-sided dominating set, and set cover, are equivalent in the following sense (see e.g., [29]).

PROPOSITION 12 (see [29]). *There is a polynomial-time reduction between any of the three problems, dominating set, one-sided dominating set, and set cover, that preserves the value of the minimum solution.*

Proof. To reduce dominating set to set cover, let V (the set of vertices) become U (the ground set) and let the collection $\mathcal{S}$ include the sets $N^+(v)$ for every $v \in V$. To reduce set cover to one-sided dominating set, construct a bipartite graph $G(V_1, V_2, E)$ in which V_2 is the ground set U and the vertices of V_1 each represent a set in $\mathcal{S}$. Put an edge $(u, v) \in E$ if $v \in V_2$ corresponds to an item that is contained in the set that corresponds to the vertex $u \in V_1$. Finally, to reduce one-sided dominating set to dominating set make a clique out of all vertices of V_1 and let $V = V_1 \bigcup V_2$; it is not hard to see that these reductions reserve feasibility of solutions, and in addition it is not hard to see that the size of the minimum dominating set is preserved. □

For set cover, let $n = |U|$. It is known that minimum set cover can be approximated within a ratio of $\ln n$ [18, 22] and that, for every fixed $\epsilon > 0$, it is hard to

approximate it within a ratio of $(1-\epsilon)\ln n$ [9]. By Proposition 12, this implies a similar result for one-sided dominating set, with $n = |V_2|$. For dominating set, it is natural to take $n = |V|$. Then the approximation ratio of $\ln n$ trivially applies, but in order to transfer the $(1-\epsilon)\ln n$ hardness result from one-sided dominating set to dominating set using the reduction of Proposition 12 we also need that $\ln|V| \simeq \ln|V_2|$, which holds whenever $|V_1| \leq |V_2|^{1+\epsilon}$. It turns out that this is indeed the case in the construction of [9]. Hence it is known that up to low order terms, $\ln n$ is the best possible approximation ratio for all three problems.

A proof similar to that of Proposition 12 shows that there is a polynomial-time reduction from domatic number (whether one-sided or not) to set cover packing number that preserves the value of the maximum solution. Likewise, there is a polynomial-time reduction from set cover packing number to one-sided domatic number that preserves the value of the maximum solution. However, we are not aware of such a reduction from one-sided domatic number to domatic number. Instead, we use the following proposition.

PROPOSITION 13. *For every integer $k > 0$ (where k may be an arbitrary function bounded by a polynomial in the size of the input) there is a polynomial-time transformation mapping a bipartite graph (V_1, V_2, E) to a graph (V, E') with the following properties:*

- $|V| = |V_2| + k|V_1|$.
- *For the original graph, let q and r, respectively, denote the one-sided domatic number and the minimum cardinality of a one-sided dominating set; hence $q \leq |V_1|/r$. Let p denote the domatic number of the new graph. Then $kq \leq p \leq \min[|V|/r, 1 + 2k|V_1|/r]$.*

Proof. Let $G = (V_1, V_2, E)$ be a bipartite graph for which we are interested in computing the one-sided domatic number. Construct a graph $G'(V, E')$ as follows. $V_1' = \bigcup_{i=1}^{k} V_1^i$, where for every i, V_1^i is a copy of V_1. $V = V_1' \bigcup V_2$, which gives $|V| = |V_2| + k|V_1|$. For every $v \in V_2$, $1 \leq i \leq k$, and $u \in V_1^i$, place an edge $(u, v) \in E'$ iff there is an edge $(u, v) \in E$. In addition, all vertices of V_1' form a clique.

Observe that every one-sided dominating set in G corresponds in a natural way to k mutually disjoint dominating sets in G', one on each copy of V_1^i. Hence the optimal solution for G can be copied k times on G', giving $p \geq kq$. (If, in addition, V_1 has no isolated vertices, then V_2 can serve as one more dominating set disjoint from all the others, giving $p \geq kq + 1$.)

To upper bound p, let $D_1, \ldots, D_p$ be a maximum cardinality collection of disjoint dominating sets in G'. Similar to the proof of Proposition 12, we can see that we can change maximum dominating set D_j to maximum dominating set D_j' of no larger size, fully contained in V_1'. As D_j' must dominate V_2, we get that $|D_j| \geq |D_j'| \geq r$ for all j. Hence $p \leq |V|/r$.

We now turn to proving the second part of the upper bound, namely, $p \leq 1 + 2k|V_1|/r$. This gives a tighter bound when $|V_2| \gg k|V_1|$. Observe that $(D_i \bigcup D_j) \bigcap V_1'$ is a dominating set contained in V_1'. So suppose we pair up any $2\lfloor p/2 \rfloor$ of the D_i's into $\lfloor p/2 \rfloor$ pairs, each of which forms a dominating set of size at least r. Then, $r\lfloor p/2 \rfloor \leq k|V_1|$; so, $p \leq 1 + 2k|V_1|/r$. □

We are now ready to prove Proposition 10.

Proof. Perform the reduction of Proposition 13 with $k = |V_2|$. If the first case holds, then the domatic number of G' is at most $|V|/r$. If the second case holds, then the domatic number is at least $q|V_2|$. The ratio between these two values is $q|V_2|/(|V|/r) = qr|V_2|/|V|$. We use $|V_2|/|V| = 1/(|V_1| + 1)$ and $qr \geq (1-\epsilon)|V_1|\ln|V_2|$

to obtain that the ratio is at least $(1 - \epsilon - 1/|V_1|) \ln |V_2|$. Using $|V_2| > |V_1|^c$ and $|V_1| \geq 4^c$, we obtain that $\ln |V_2| \geq (1 - 1/(c + 1/2)) \ln |V|$. Now Proposition 10 follows assuming that $|V_1| \geq 2(c + 1)^2$. □

Remark. The graphs G' formed in the above reduction are *split graphs*, which are graphs whose vertex-set can be partitioned into a clique and an independent set. These graphs are both chordal and complements of chordal graphs, where a graph is chordal if it contains no cycle with four or more vertices as an induced subgraph. Thus, a $(1 - o(1)) \ln n$ hardness for domatic number holds also for split (and thus chordal and cochordal) graphs. Furthermore, one can get a similar result for bipartite graphs by the following modification. Instead of making a clique out of V_1, we add $(\delta+1)$ vertices and connect them to every node in V_1. This affects the domatic number by at most 1, and the approximability hardness follows.

3.3. The problem Max 3-colorability-5. We now introduce the NP-language that serves as the basis to our reduction.

DEFINITION 2. Max 3-colorability *is the problem of coloring the vertices of a graph with three colors so as to maximize the number of legally colored edges (edges whose endpoints are colored differently).*

An NP-witness for 3-colorability is an assignment of colors to the vertices such that each edge is legally colored. The witness can be checked in a probabilistic sense by sampling an edge at random and checking whether the colors of its two endpoints disagree. The actual names of the colors of the vertices play no role because the names of the three colors can be arbitrarily permuted without changing the legality of the 3-coloring. Hence every NP-witness gives rise to six different witnesses (depending on the permutation used for the names of the colors), and cycling over the six witnesses, every edge gets its six legal colorings. In our reductions to one-sided dominating set, we shall use this structure of the set of witnesses for 3-colorability in order to show that if the resulting bipartite graph has a small one-sided dominating set, then in fact it has many disjoint one-sided dominating sets. We note that the same structure was used in [15] to construct a zero-knowledge proof system for NP.

In order to prove a hardness result for Max 3-colorability-5, we rely on the hardness of Max 3-colorability and the use of expanders. Call a graph H with h vertices a (γ, κ)-*expander* if for any subset S of the vertices of H with $|S| \leq \gamma h$, the number of edges leaving S is at least $\kappa |S|$. For some constant $\kappa > 0$ and for any $\epsilon > 0$, there is an h_0 such that for all $h \geq h_0$, there is an explicitly constructible $(1/2, \kappa)$-expander with maximum degree at most 6 and with the number of vertices lying in the range $[h, h(1 + \epsilon)]$ [23]. Note that a $(1/2, \kappa)$-expander is necessarily also a $(2/3, \kappa/2)$-expander. In particular, this implies that for any given integer h, we can construct a $(2/3, \kappa')$-expander with maximum degree at most 6 and with the number of vertices being some $F(h)$ that satisfies $h \leq F(h) \leq 2h$ for some absolute constant $\kappa' > 0$. (We can proceed as follows. If $h \leq h_0$, where h_0 is a sufficiently large constant, just construct any connected h-vertex graph of maximum degree 6. If $h > h_0$, construct a $(1/2, \kappa)$-expander with maximum degree at most 6 and with the number of vertices lying in the range $[h, 2h]$, using [23].)

We will also need the following theorem of [31].

THEOREM 14 (see [31]). *For some explicit constant $\psi < 1$, it is NP-hard to distinguish between graphs that have a legal 3-coloring (that colors all edges legally), and graphs for which every 3-coloring legally colors at most a ψ-fraction of the edges.*

PROPOSITION 15. *For some explicit constant $\psi < 1$, it is NP-hard to distinguish between 5-regular graphs that have a legal 3-coloring, and 5-regular graphs for which*

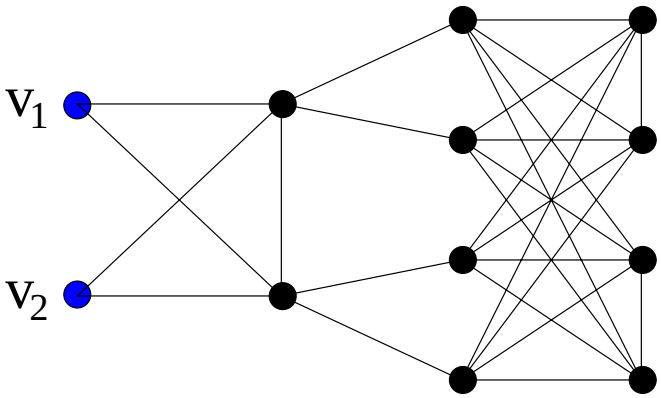


FIG. 1. *Equality gadget.*

every 3-coloring legally colors at most a ψ-fraction of the edges.

Proof. Given a graph $G(V, E)$, we show that it can be modified in polynomial-time to a graph $G'(V', E')$ such that (i) G' is 5-regular; (ii) $|E'| = O(|E|)$; (iii) G' is legally 3-colorable iff G is; and (iv) for some constant $c > 0$ and every $1 \leq k \leq |E'|$, every 3-coloring of G' that leaves k edges illegally colored can be transformed in polynomial-time to a 3-coloring of G that leaves at most ck illegally colored edges. We can then invoke Theorem 14.

First, we may assume that every vertex in G has degree at least three. Indeed, suppose we repeatedly remove any vertex of degree at most 2 until the remaining graph G'' has minimum degree at least 3. It is easy to see that, given any 3-coloring of the vertices of G'', we can add back the deleted vertices of G and color these added-back vertices in such a way that all the edges incident with them are legally colored. So, we assume that G has minimum degree at least 3.

Assume first that all vertices in G have degree at most 13. Then we replace each vertex v by a cluster of $d(v)$ vertices,[1] where each vertex handles one outgoing edge. (That is, if (v, u) is an edge in G, then the vertex representing u in v's cluster is made adjacent to the vertex representing v in u's cluster.) These $d(v)$ vertices are connected in a cycle by *equality gadgets*, shown in Figure 1. These equality gadgets are subgraphs that contain twelve vertices. Two of the vertices are special and have degree two, and all the rest have degree five. The equality gadget has the property that it is legally 3-colorable iff the special vertices receive the same color. The two cluster vertices serve as the special vertices. As each cluster vertex participates in two gadgets and also has one outgoing edge, its degree is 5. Hence G' is 5-regular. The number of edges added is at most $27 \sum_v d(v) \leq 54|E|$, and hence, $|E'| \leq 55|E|$. Every legal 3-coloring of G' colors all cluster vertices with the same color and hence naturally gives a legal 3-coloring of G. Similarly, a legal 3-coloring of G can be extended to a legal 3-coloring of G'. Any 3-coloring of G' in which cluster vertices receive different colors causes at least two edges of G' to be miscolored. Coloring the vertex corresponding to the cluster with an arbitrary color in G causes at most $d(v) \leq 13$ edges to be miscolored. Hence the number of illegally colored edges in G' is smaller than that of G by a factor of at most 13/2.

If G has a vertex of degree more than 13, we create a new graph G_1 as follows.

[1] Recall that $d(v)$ denotes the degree of vertex v [36].

Let $F(\cdot)$ and κ' be as in our discussion on expanders above. Similarly as [28], G_1 is the same as the G' constructed in the previous paragraph, except that for each vertex v, we create a $(2/3, \kappa')$-expander with $F(d(v))$ vertices and maximum degree at most 6 (instead of a cycle with $d(v)$ vertices) and do the above-seen operation of replacing the expander edges by equality gadgets. (Of the $F(d(v))$ vertices, $d(v)$ are "real" and represent the neighbors of v, and $F(d(v)) - d(v)$ are dummy vertices. As in the previous paragraph, if (v, u) is an edge in G, then the vertex representing u in v's cluster is made adjacent to the vertex representing v in u's cluster.) Note that G_1 has maximum degree at most $2 \times 6 + 1 = 13$ and has $O(\sum_v d(v)) = O(|E|)$ edges. Once again, a 3-coloring of G can be efficiently transformed into one for G_1 in which the number of miscolored edges remains the same. Conversely, suppose we have a 3-coloring χ_1 of G_1. Let X_1 be the number of original edges of G that are miscolored by χ_1 and let Y_1 be the number of edges in the equality gadgets miscolored by χ_1. We will produce the following coloring χ of G and then analyze χ. For each vertex v, choose a largest subcluster $C(v)$ of vertices (from its cluster of $F(d(v))$ vertices) that receive the same color in χ_1 and define $\chi(v)$ to be the color assigned to the vertices in $C(v)$ by χ_1. Call an edge (u, v) of G *bad* if it was properly colored by χ_1 in G_1 and was miscolored by χ in G. Let X be the number of bad edges. Then the number of edges miscolored by χ is at most $X_1 + X$. An edge (u, v) is bad only if "$u \notin C(v)$ or $v \notin C(u)$" holds. Thus, letting $x(v) = F(d(v)) - |C(v)| \leq 2F(d(v))/3$, we see that $X \leq \sum_v x(v)$. Since $x(v) \leq 2F(d(v))/3$ and recalling the property of $(2/3, \kappa')$-expansion, we can check that the number of edges in all the equality gadgets of v that are miscolored in χ_1 is $\Omega(x(v))$. Hence $Y_1 = \Omega(X)$. Armed with this property and the fact that G_1 has $O(|E|)$ edges and maximum degree 13, we transform G_1 into G' as described in the previous paragraph. □

3.4. Preliminaries. Our reduction closely follows that of [9]. The main differences are as follows: (i) The outcome of the reduction is one-sided dominating set, rather than set cover (which is the same thing termed differently). (ii) The starting point of the reduction is Max 3-colorability rather than Max 3SAT. As mentioned earlier, the purpose of this change is to have the reduction apply to one-sided domatic number rather than just one-sided dominating set. (iii) Every vertex in the V_1 side of the bipartite graph will be duplicated $2^{l/2}$ times, where l is a parameter of the reduction. This is a technical condition that seems to be required when we reason about the one-sided domatic number.

To describe the reduction, we recall two notions used in [9].

DEFINITION 3. *A* (k, l)-Hadamard code *is a set of* k *binary words of length* l*, where every codeword has Hamming weight* $l/2$ *and the Hamming distance between every two codewords is* $l/2$.

There is a simple construction of Hadamard codes when l is a power of 2 and $k \leq l$ (see, e.g., [25]).

DEFINITION 4. *A* partition system $B(m, L, k, d)$ *has the following properties:*

- *There is a ground set* B *of* m *points.*
- *There is a collection of* L *distinct* partitions $p_1, \ldots, p_L$.
- *For* $1 \leq i \leq L$*, partition* p_i *is a collection of* k *disjoint subsets whose union is* B.
- *Any cover of the* m *points by subsets that appear in pairwise different partitions requires at least* d *subsets.*

The following lemma is proved in [9].

LEMMA 16. *For every* $c \geq 0$ *and* m *sufficiently large there is a* partition system

$B(m, L, k, d)$ whose parameters satisfy the following:

- $L \simeq (\log m)^c$.
- *k can be chosen arbitrarily as long as* $k < \frac{\ln m}{3 \ln \ln m}$.
- $d = (1 - f(k))k \ln m$, *where* $f(k) \to 0$ *as* $k \to \infty$.

Moreover, a random collection of L partitions into k subsets with parameters chosen as above gives with high probability a partition system with $f(k) = 2/k$.

The randomized construction can be replaced by a deterministic construction (with a somewhat larger value for $f(k)$) using techniques developed in [27].

3.5. The construction. The input to the reduction is a 5-regular graph $G(V, E)$ for which we want to determine whether it is legally 3-colorable, or whether every 3-coloring of its vertices legally colors at most $\psi|E|$ edges. As noted in Proposition 15, for some explicit $\psi < 1$ this problem is NP-hard. The reduction uses the following parameters, which are chosen so that (k, l)-Hadamard codes exist and Lemma 16 holds:

- $k = l = c \log \log |V|$ for some sufficiently large constant c. We assume that l is a power of 2.
- $L = 3^l$, and $m = |V|^{\Theta(l)}$.

The output of the reduction is a bipartite graph $G'(V_1, V_2, E')$; when we say "color" below, we refer to a color-set of 3 colors. The left-hand side vertex-set V_2 is composed of $(2|E|)^l$ clusters of vertices, where each cluster contains m vertices. Each left-hand side cluster is labeled by a sequence of l edges in G and a sequence of l bits; for each i, the ith bit in the bit-sequence denotes one endpoint (vertex) of the ith edge in the edge-sequence. Hence, this sequence of bits can also be viewed as a sequence of vertices. The right-hand side vertex-set V_1 is composed of k disjoint *rays*, and each ray is labeled by a codeword of the (k, l)-Hadamard code. Each ray is composed of $|V|^{l/2}|E|^{l/2}$ clusters, where each cluster contains 6^l vertices. Each right-hand side cluster is labeled by a sequence of $l/2$ vertices in G and a sequence of $l/2$ edges in G. Equivalently, we may merge these two sequences to one sequence of length l, where the codeword of the ray containing the cluster is used as a selector function specifying the order in which vertices and edges are merged (a vertex when the corresponding bit in the codeword is 0, and an edge when the corresponding bit in the codeword is 1). This is called the merged label of a right-hand side cluster. Individual vertices of right-hand side clusters are further labeled by a sequence of $l/2$ colors (i.e., a ternary sequence), a sequence of $l/2$ pairs of distinct colors (i.e., a sequence in base 6), and a number between 1 and $2^{l/2}$. Simple counting shows that for each of the labeling schemes that we defined, the number of available labels is exactly equal to the number of objects that need to be labeled.

We say that a left-hand side cluster and a right-hand side cluster are *compatible* if their labels agree coordinate-wise in the following sense: for coordinate i, if the merged label of the right-hand side cluster has an edge, then this is the ith edge in the sequence of edges labeling the left-hand side cluster, and if the merged label has a vertex, then this is the ith vertex in the sequence of vertices labeling the left-hand side cluster. Edges in G' only connect compatible clusters (compatibility is necessary but not sufficient, as will be seen shortly). Note that each left-hand side cluster is compatible with exactly one cluster in each ray. Each right-hand side cluster is compatible with $5^{l/2}2^{l/2}$ left-hand side clusters; the term "5" here arises from the fact that G is 5-regular, and the term "2" follows from the fact that every edge has two end-points.

In order to describe the edge set E', we use the notion of a partition system. For each left-hand side cluster C_ℓ, we have $L = 3^l$ partitions with properties as in Definition 4. (Recall that C_ℓ has m elements as required by Definition 4.) Each partition is labeled by a sequence of l colors. This sequence of colors is interpreted as a sequence of colors for the sequence of vertices that label C_ℓ. Note that the same vertex of G may appear several times in the sequence of vertices that labels C_ℓ; we do not require the colors given to this vertex to be the same.

Consider an arbitrary vertex v in a cluster C_r that belongs to ray i. We now describe the set of neighbors that it has in a *compatible* left-hand side cluster C_ℓ. Recall that v is labeled by a sequence of $l/2$ colors and a sequence of $l/2$ pairs of distinct colors. These colors give in a natural way a coloring for the merged sequence of vertices and edges labeling C_r. The vertex v was also labeled by a number between 1 and $2^{l/2}$; this label is ignored when determining the set of neighbors of v (that is, C_r has $2^{l/2}$ identical copies of v).

The coloring of the merged sequence of C_r induces in a natural way a coloring for the sequence of vertices labeling C_ℓ. This coloring labels one particular partition p. Vertex v is connected to all vertices (points) of the ith part of partition p (recall that i is the ray to which C_r belongs). This completes the description of E'.

LEMMA 17. *If G is legally 3-colorable, then the one-sided domatic number of G' is 6^l.*

Proof. We first show that G' has a one-sided dominating set that includes exactly one vertex from every right-hand side cluster. Consider a sequence of l arbitrary legal 3-colorings of G (the same legal 3-coloring may appear multiple times in the sequence). Now consider an arbitrary right-hand side cluster. It is labeled by a length l merged sequence of vertices and edges. The sequence of legal 3-colorings induces a coloring on this merged sequence. The cluster contains exactly $2^{l/2}$ vertices whose label induces the same coloring. Select one of them arbitrarily to be included in the one-sided dominating set.

To show that indeed we have a one-sided dominating set, we need to show that every left-hand side vertex u is covered. Consider the cluster C_ℓ to which u belongs. It is labeled by a sequence of l edges and a sequence of l vertices. The sequence of legal 3-colorings induces a coloring for the sequence of vertices. This coloring agrees with the name of one partition p. Let i be the part of partition p to which u belongs. Consider the right-hand side cluster C_r that is compatible with C_ℓ and belongs to ray i. The vertex selected from C_r necessarily covers u.

We now show that the one-sided domatic number is at least 6^l (in fact, it is exactly 6^l). Consider an arbitrary legal 3-coloring of G. From it, we can derive 6^l distinct length l sequences of legal 3-colorings, where in each of the l coordinates we put one of the six permutations of the legal 3-coloring. Each of these sequences gives a one-sided dominating set as described above. We show that these one-sided dominating sets can be chosen to be distinct. This follows from the fact that for each sequence of legal 3-colorings and every right-hand side cluster, we can have $6^{l/2}3^{l/2}$ equivalence classes of $2^{l/2}$ vertices (who differ only in the number they are given in the third label) and $6^{l/2}3^{l/2}$ equivalence classes of $2^{l/2}$ sequences of colorings (who differ only in the way they color vertices not in the merged sequence of the right-hand side cluster). Preserving the structure of the equivalent classes, we can match the sequences of legal 3-colorings with the vertices of a right-hand side cluster. □

LEMMA 18. *If every 3-coloring of G legally colors at most $\psi|E|$ edges, then the smallest one-sided dominating set in G' is of cardinality at least $(1-o(1))|V_1| \ln m/6^l$.*

Proof. The proof is essentially identical to that of Lemma 7 in [9] and is omitted. □

By making m sufficiently large, we can have $|V_2| > |V_1|$ and $\ln m \simeq \ln |V_2|$, and the proof of Theorem 11 follows.

Appendix A. Multicoloring version. In the domatic multipartition problem, we are additionally given an integral weight $x : V \mapsto N$ indicating in how many dominating sets each vertex can appear. The domatic number problem has $x(v) = 1$ for each v; the *r-Conf* problem of [13] has $x(v) = r$ for each v.

We can reduce the multipartition problem to the ordinary partition problem. Given a graph G and weight vector x, form a graph G' as follows. G' has $x(v)$ copies of each vertex v connected as a clique. For each edge uv in G, the copies of u and v form a complete bipartite graph in G'. Any minimal dominating set in G' is also a dominating set in G, and taking one copy of each vertex of a dominating set in G also gives a dominating set in G'. Further, a domatic partition of G' is in one-to-one correspondence with a multipartition of G (within the weight constraints). Thus, the results obtained in this paper for the domatic number problem carry over to the domatic multipartition problem, replacing δ by $\min_v d(v)x(v)$ and n by $\sum_v x(v)$.

Appendix B. Overview of some ingredients from [9].

Parallel repetition of two-prover proof systems. There is a straightforward one-round two-prover proof system for Max 3-colorability-5 that has the following properties: on *yes* instances, the verifier always accepts, and on *no* instances (when at most a $(1 - \epsilon)$-fraction of the edges can be legally colored simultaneously) the verifier accepts with probability at most $1 - \epsilon/3$. Give a 5-regular graph G, the proof system proceeds as follows. The verifier sends to the first prover a random edge in G and to the second prover a random vertex from that edge. We call this vertex the *common* vertex. It is important that the first prover does not know which endpoint of the edge is the common vertex and that the second prover does not know which edge was received by the first prover. The first prover replies with two different colors (out of the three allowable colors) for the two vertices that are the endpoints of the edge. The second prover replies with a color for the common vertex. The verifier accepts only if the two provers give the same color to the common vertex. For a *yes* instance G the two provers can answer according to the global legal 3-coloring and ensure that the verifier accepts with probability 1. For a *no* instance G, regardless of the strategy of each prover (where a strategy is a function from questions to answers), the acceptance probability is at most $1 - \epsilon/2$, where probability is computed over the random choices of the verifier.

The l-fold parallel repetition of this one-round two-prover proof system is as follows. The verifier sends to the first prover a tuple of l random edges and to the second prover a tuple of l random vertices, one from each of these edges. Each prover replies with colors to all the vertices that it receives. The verifier accepts if on every one of the l common vertices, the two provers agree on the color that they give. It is not hard to see that on *yes* instances, the provers still have a strategy that makes the verifier accept with probability 1. The parallel repetition theorem [34] shows that on *no* instances, the verifier accepts with probability at most $(1 - \epsilon)^{cl}$, where c is a constant (that depends on ϵ).

Separating codes and *k*-provers. To eventually get $\simeq \ln n$ hardness of approximation results, two-prover proof systems are extended to k-prover systems in a special way. As in the two-prover proof system, the verifier selects l edges at random and a random vertex within each edge. This gives a total of $(2|E|)^l$ possible queries.

Each of the k provers is sent some subset of the l edges and l vertices using the following rule. A (k, l)-Hadamard code is a collection of k binary words of length l each, for which the Hamming distance between any pair of words is "large" (specifically, $l/2$). With each prover we associate one codeword from the Hadamard code. The verifier sends to each prover $l/2$ edges and $l/2$ vertices, selected according to the 1's and 0's in the codeword of the respective prover. Namely, if the ith bit in the code is 1, the next entry in the query is the respective ith edge (among the l edges in the tuple). If the ith entry is 0, the ith vertex in the vertex-tuple is sent. Considering separately two of the provers, the fact that their codewords have a large Hamming distance implies that in many coordinates one prover will receive an edge while the other prover will receive a vertex on this edge. This is similar to the scenario in the two-prover proof system. Intuitively, we may view the k-prover proof system as $\binom{k}{2}$ correlated two-prover proof systems going on in parallel. It is natural to have the verifier accept if all $\binom{k}{2}$ proof systems are accepting. The definition of acceptance used in [9] is more subtle and will not be discussed in this overview.

Next a bipartite graph $G'(V_1, V_2, E')$ is built based on this scenario. Specifically, the vertices in V_1 describe the provers/queries schema. The set V_1 corresponds to the provers and is partitioned into k "rays" V_1^j, $1 \le j \le k$, one ray for each prover. The sets V_1^k are further partitioned into disjoint subsets $V_1^j(Q)$, where Q ranges over all the $|V|^{l/2}|E|^{l/2}$ possible questions that a prover can receive. On each possible question, the prover must answer with a coloring of the $l/2$ edges and $l/2$ vertices received. Thus, the $V_1^j(Q)$ sets are further divided into all possible answers. Therefore, there are $3^{l/2} \cdot 6^{l/2}$ points inside $V_1^j(Q)$ in this final division; a point for each possible coloring of the $l/2$ vertices and $l/2$ edges. In addition, for technical reasons, each point (possible answer) of each prover is duplicated $2^{l/2}$ times.

Ground sets and random partitions. To determine the structure of V_2, some ideas essentially due to [24] (and extended in [9]) are used.

The set V_2 is partitioned into $(2|E|)^l$ *ground sets* C_ℓ, one ground set per each possible query. Each ground set contains m points for $m = |V|^{\Theta(l)}$. It is instructive to note the following asymmetry. Given a query, namely, a sequence of l edges and l corresponding endpoints for the edges, consider its corresponding ground set in V_2. There are exactly k sets $V_1^j(Q)$ that are compatible with this sequence, one per prover (since the question to the provers is completely determined by the Hadamard code). On the other hand, given a question cluster $V_1^j(Q)$, many possible query-base ground sets in V_2 could have caused this question. Indeed, since the question contains only $l/2$ edges and $l/2$ vertices, this partial information can be completed in $5^{l/2}2^{l/2}$ ways to give compatible l edges and l vertices. Each vertex sent to this prover has 5 neighbors and thus 5 ways of "completing" this vertex into a (compatible) edge. For each edge sent to this prover, there are 2 possibilities for which of the two vertices to put in the resulting query.

We use $L = 3^l$ random partitions of C_ℓ, each partitioning C_ℓ into k parts. Consider the l-tuple of vertices in a possible query (namely, the part of the query corresponding to the chosen vertices, one per each edge). Note that the number $L = 3^l$ of partitions of each ground set exactly equals the number of possible colorings 3^l of the vertices in that query. We can thus form a 1-1 correspondence between the 3^l partitionings and the 3-coloring (not necessarily a consistent one) of the l vertices in the query base. Each partition is now regarded as a coloring and vice versa. Recall that a point in the final subpartition cluster of V_1 corresponds to an answer to the coloring of the query of $l/2$ edge/vertex pairs. This corresponds to a coloring of the

l vertices of the cluster C_ℓ and hence gives a vertex coloring, and thus a partition of C_ℓ. The edges between the vertices of V_1 and V_2 are defined as follows. Each vertex (namely, coloring or answer) $v \in V_1^j(Q)$ is connected to the jth part of the partition corresponding to v (as v is a coloring, a partition is also associated with v).

Now, consider the graph G' resulting from a 3-colorable graph G. We can choose one point (answer) per each question-cluster in V_1, so as to cover each C_ℓ. Thus all the C_ℓ are covered using at most $k \cdot (2 \cdot E)^l$ vertices. This will happen if the provers use strategies which are all part of a global 3-coloring, and this gives a small dominating set. On the other hand, consider the resulting graph G' when only $1 - \epsilon$ of the edges of G can be simultaneously legally colored. Because of the parallel repetition which greatly increases the gap, the provers essentially cannot use a joint strategy. This means that each pair of answers $v_j \in V_1^j(Q), v_q \in V_1^q(Q)$ chosen corresponds to two different colorings. Hence a ground set C_ℓ is essentially covered via random sets each containing m/k random elements of C_ℓ. Thus, the number of elements needed in order to cover C_ℓ is roughly $k \ln m$. This gives a total of roughly $k \cdot (2 \cdot E)^l \ln |V'|$ minimum one-sided dominating sets for a logarithmic gap.

Zero-knowledge and the domatic number. Given the fact that a single "small" one-sided dominating set exists, we can permute the colors (as done in zero-knowledge protocols in order to "hide" information) to show the existence of "many" disjoint dominating sets, hence a "large" domatic number. Namely, once a 3-coloring of G exists, one gets 6 legal colorings for G. This implies a total of 6^l colorings (which can be completed into global coordinatewise-compatible colorings) of any l-tuple of vertices. The possibility for color permutation was the main reason for reducing from a coloring and not a satisfiability problem. Hence, we either get many small dominating sets which implies a large one-sided domatic number, or any dominating set is large, which gives a small one-sided domatic number.

Acknowledgments. We would like to thank Satoshi Fujita and Mario Szegedy for helpful discussions and the two referees for their helpful comments. The first author is the incumbent of the Joseph and Celia Reskin Career Development Chair. The second author would like to thank Kazuo Iwama at Kyoto University for his hospitality.

REFERENCES

[1] N. Alon and J. Spencer, *The Probabilistic Method*, John Wiley, New York, 1992.
[2] S. Arnborg, J. Lagergren, and D. Seese, *Easy problems for tree-decomposable graphs*, J. Algorithms, 12 (1991), pp. 308–340.
[3] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and intractability of approximation problems*, J. ACM, 45 (1998), pp. 501–555.
[4] J. Beck, *An algorithmic approach to the Lovász Local Lemma*, Random Structures Algorithms, 2 (1991), pp. 343–365.
[5] C. Berge, *Balanced matrices*, Math. Program., 2 (1972), pp. 19–31.
[6] M. A. Bonucelli, *Dominating sets and domatic number of circular arc graphs*, Discrete Appl. Math., 12 (1985), pp. 203–213.
[7] P. Erdős and L. Lovász, *Problems and results on 3-chromatic hypergraphs and some related questions*, in Infinite and Finite Sets, Vol. II, Colloq. Math. Soc. János Bolyai 11, A. Hajnal et al., eds., North–Holland, Amsterdam, 1975, pp. 609–627.
[8] M. Farber, *Domination, independent domination, and duality in strongly chordal graphs*, Discrete Appl. Math., 7 (1984), pp. 115–130.
[9] U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.
[10] U. Feige, M. M. Halldórsson, and G. Kortsarz, *Approximating the domatic number*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 134–143.

[11] U. Feige and J. Kilian, *Zero knowledge and the chromatic number*, J. Comput. System Sci., 57 (1998), pp. 187–199.
[12] S. Fujita, *On the performance of greedy algorithms for finding maximum r-configurations*, in Proceedings of Korea-Japan Joint Workshop on Algorithms and Computation (WAAC), Seoul National University, Seoul, 1999, pp. 92–99.
[13] S. Fujita, M. Yamashita, and T. Kameda, *A study on r-configurations—a resource assignment problem on graphs*, SIAM J. Discrete Math., 13 (2000), pp. 227–254.
[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
[15] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proofs*, J. ACM, 38 (1991), pp. 691–729.
[16] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, eds., *Domination in Graphs: Advanced Topics*, Marcel Dekker, New York, 1998.
[17] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of Domination in Graphs*, Marcel Dekker, New York, 1998.
[18] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.
[19] H. Kaplan and R. Shamir, *The domatic number problem on some perfect graph families*, Inform. Process. Lett., 49 (1994), pp. 51–56.
[20] S. Khanna, M. Sudan, and D. Williamson, *A complete classification of the approximability of maximization problems derived from Boolean constraint satisfaction*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 11–20.
[21] J. Kratochvíl, *Regular codes in regular graphs are difficult*, Discrete Math., 133 (1994), pp. 191–205.
[22] L. Lovász, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.
[23] A. Lubotzky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica, 8 (1988), pp. 261–277.
[24] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41 (1994), pp. 960–981.
[25] F. MacWilliams and N. Sloane, *The Theory of Error Correcting Codes*, North–Holland, Amsterdam, 1983.
[26] M. V. Marathe, H. B. Hunt, III, and S. S. Ravi, *Efficient approximation algorithms for domatic partition and on-line coloring of circular arc graphs*, Discrete Appl. Math., 64 (1996), pp. 135–149.
[27] M. Naor, L. Schulman, and A. Srinivasan, *Splitters and near-optimal derandomization*, in Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, 1995, pp. 182–191.
[28] C. H. Papadimitriou and Y. Yannakakis, *Optimization approximation and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.
[29] A. Paz and S. Moran, *Nondeterministic polynomial optimization problems and their approximations*, Theoret. Comput. Sci., 15 (1981), pp. 251–277.
[30] S. L. Peng and M. S. Chang, *A simple linear time algorithm for the domatic partition problem on strongly chordal graphs*, Inform. Process. Lett., 43 (1992), pp. 297–300.
[31] E. Petrank, *The hardness of approximation: Gap location*, Comput. Complexity, 4 (1994), pp. 133–157.
[32] A. Proskurowski and J. A. Telle, *Complexity of graph covering problems*, Nordic J. Comput., 5 (1998), pp. 173–195.
[33] P. Raghavan, *Probabilistic construction of deterministic algorithms: Approximating packing integer programs*, J. Comput. System Sci., 37 (1988), pp. 130–143.
[34] R. Raz, *A parallel repetition theorem*, SIAM J. Comput., 27 (1998), pp. 763–803.
[35] A. S. Rao and C. P. Rangan, *Linear algorithm for domatic number problem on interval graphs*, Inform. Process. Lett., 33 (1989), pp. 29–33.
[36] A. Srinivasan, *Domatic partitions and the Lovász local lemma*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2001, pp. 922–923.
[37] B. Zelinka, *Domatic number and degree of vertices of a graph*, Math. Slovaca, 33 (1983), pp. 145–147.

SIAM J. COMPUT.
Vol. 32, No. 1, pp. 196–230

MACHINE-INDEPENDENT CHARACTERIZATIONS AND COMPLETE PROBLEMS FOR DETERMINISTIC LINEAR TIME*

ETIENNE GRANDJEAN† AND THOMAS SCHWENTICK‡

Abstract. This article presents two algebraic characterizations and two related complete problems for the complexity class **DLIN** that was introduced in [E. Grandjean, Ann. Math. Artif. Intell., 16 (1996), pp. 183–236]. **DLIN** is essentially the class of all functions that can be computed in linear time on a Random Access Machine which uses only numbers of linear value during its computations.

The algebraic characterizations are in terms of recursion schemes that define unary functions. One of these schemes defines several functions simultaneously, while the other one defines only one function. From the algebraic characterizations, we derive two complete problems for **DLIN** under new, very strict, and machine-independent *affine* reductions.

Key words. linear time, model of computation, Random Access Machine, completeness, recursion schemes, affine reductions

AMS subject classifications. 68Q, 03D

PII. S0097539799360240

1. Introduction.

In the definition of the main complexity classes like deterministic or nondeterministic polynomial time or polynomial space, the choice of the underlying machine model is not very crucial. The definition does not depend on whether one chooses 1-tape, multitape, or multidimensional Turing machines or RAMs. As long as the cost associated with a computation is reasonable, all definitions define the same classes—with the notable exception of the nonreasonable unit-cost model for RAMs that are allowed to multiply numbers. Linear time, on the other hand, is a much more delicate notion. For instance, as in algorithm design, it is very sensitive to the representation of inputs and to changes of the computational model. It is therefore a nontrivial task to define a robust notion of linear time. Some authors tried to circumvent this problem by considering so-called *quasi-linear time*, i.e., time $O(n\text{polylog}(n))$ [33, 23, 12]. On the other hand, there is even no general agreement on whether linear time on Turing machines is too weak or too powerful [31].

The underlying formalization of deterministic linear time for this article, the class **DLIN**, was defined and investigated by one of the authors in a series of articles [18, 17, 19]. Before we recall some details of this definition, let us first have a closer look at why deterministic linear time on Turing machines has little to do with linear time algorithms on, say, graphs. First, such algorithms rely usually on more sophisticated representations of the input graphs, such as adjacency lists. Such lists are basically pointer structures, and the algorithms usually make intensive use of pointer variables. As the movements of the heads of a Turing machine are only local, it is hard to see how such "pointer jumping" algorithms could be simulated on a Turing machine in linear time. On the other hand, RAMs are perfectly suited to perform such algorithms. The definition of the class **DLIN** for string problems is based on the idea of representing

*Received by the editors August 13, 1999; accepted for publication (in revised form) February 12, 2002; published electronically December 11, 2002. This research was supported by PROCOPE.
http://www.siam.org/journals/sicomp/32-1/36024.html

†Dépt. d'Informatique, Université de Caen, GREYC, 14032 CAEN Cedex, France (etienne.grandjean@info.unicaen.fr).

‡Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, 35032 Marburg, Germany (tick@informatik.uni-marburg.de). This work was done while this author was affiliated with the University of Mainz, Germany.

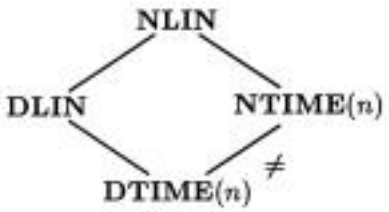


FIG. 1. *The known inclusions between the linear time classes considered.*

strings by pointer structures in analogy to efficient graph representations. In a first step, a string $w = w_1 \cdots w_n$ of length n is partitioned into $m := \lceil n/l \rceil$ pieces of length $l = \lceil \frac{1}{2} \log n \rceil$. Then such a partitioned string is encoded as a unary function $f : \{1, \ldots, m\} \to \{0, \ldots, m\}$ by defining $f(i)$ to be the number which is encoded[1] by the ith piece of the (partitioned) string w. A computation is linear time if it needs only a linear number of steps in m. The second characteristic feature of the model is that, during a computation, only numbers of value $O(m)$ are allowed. The model is quite robust with respect to arithmetic operations. We can choose addition and subtraction as the basic arithmetic operations. Allowing multiplication does not change the computational power. A complete definition is given in section 2. With **DLIN** and **NLIN** we denote the classes of problems that can be computed on such a deterministic (resp., nondeterministic) RAM in linear time. Both classes are quite robust and seem to be very reasonable formalizations of the intuitive notion of linear time.

Figure 1 shows the known inclusions between **DLIN**, **NLIN** and the linear time classes for (multitape) Turing machines, **DTIME**(n), and **NTIME**(n). The separation of **DTIME**(n) from **NTIME**(n) was shown in [30].

In [15, 16, 18, 21], it is shown that **NLIN**, the nondeterministic counterpart of **DLIN**, has a number of nice properties.

- **NLIN** contains all 21 **NP**-complete problems that were defined by Karp in his seminal paper on **NP**-completeness [25].
- **NLIN** coincides with the class of problems that can be characterized by logical formulas of the form $\exists f_1, \ldots, f_k$ for all x φ, where the f_i are unary function symbols and φ is a quantifier-free formula.
- There are natural complete sets for **NLIN** under **DTIME**(n)-reductions. These complete sets are derived from the mentioned logical characterizations. As **DTIME**$(n) \subset$ **NTIME**$(n) \subseteq$ **NLIN**, they are not in **DTIME**(n).
- **NLIN** coincides with the class of problems that can be solved in linear time on a nondeterministic Turing machine that is allowed to perform a constant number of string-sorting operations.

In a very recent paper [26], a logical characterization of **NTIME**(n) was given. Logical characterizations of deterministic linear time classes seem to be more difficult to come by. For **DLIN**, a decisive step was accomplished by one of the authors [34], who gave the first machine-independent characterization of this class and of related deterministic linear time classes.

In this article, we give algebraic characterizations of **DLIN**, from which we derive complete problems for this class under very strict reductions. These results significantly simplify and improve the characterizations that were given in [34]. We hope

[1] via the dyadic encoding which maps the string $0 \cdots 0$ to 0, $0 \cdots 01$ to 1, $0 \cdots 10$ to 2, and so on.

that these algebraic characterizations will lead to clean logical characterizations of **DLIN** which will eventually give us logical games as tools for lower bound proofs. For a discussion of the connections between linear time complexity classes and inexpressibility proofs in finite model theory, see [35].

It should be noted that there exist many characterizations of classical computational complexity classes in terms of algebraic recursion schemes in the literature, e.g., [4, 22, 23, 6, 2, 1, 14, 28, 3].

The article is organized as follows. In section 2, we give precise definitions of the underlying computational model and the class **DLIN**. Furthermore, we introduce the notion of *affine reductions*, a very strict type of reductions that are natural in the context of RAM computations. In section 3, we describe the algebraic recursion schemes that are used in the characterizations which are proved in section 4. In section 5, we exhibit complete problems for **DLIN**. We conclude with a summary of our results and possible ways they may be extended in section 6.

2. Preliminaries. For natural numbers $n > 0$, we write $[n]$ to denote the set $\{0, \ldots, n-1\}$. All arithmetic operations that we consider take only nonnegative integer values. In particular, we use the *underflow convention* that $a - b$ takes the value 0 if $a < b$.

2.1. RAM data structures. As mentioned in the introduction, the notion of linear time depends strongly on the computational model and the representation of the input. In this article, our model will be the RAM. The basic objects that a RAM manipulates in a single step are numbers, as opposed to Turing machines where the basic objects are characters of a finite alphabet. Therefore, it is reasonable to view the input and the output of a RAM as sequences of numbers, as opposed to sequences of characters, i.e., strings. It has turned out in previous work [17, 19] that one gets a robust model of linear time on RAMs if the running time, the number of registers, and the value of all numbers used during a computation, including the input numbers, are linearly bounded in the length of the input sequence, i.e., in the number of its numbers. A sequence $a_0, \ldots, a_{n-1}$ can be nicely represented by a function $f : [n] \to \mathbb{N}$, where f is defined by $f(i) = a_i$ for each $i \in [n]$. In general, we want to be able to consider inputs that consist of several sequences of numbers. This leads to our first basic definition. Let $\mathbf{t}$ be a set of function and constant symbols. A *RAM data structure s of type* $\mathbf{t}$ consists of

- a positive *size* $n \in \mathbb{N}$,
- a constant $C \in \mathbb{N}$ for each constant symbol C of $\mathbf{t}$, and
- a function $f : [n] \to \mathbb{N}$ for each function symbol f in $\mathbf{t}$.

We refer to constants C of s by $s.C$ and to functions f of s by $s.f$. We consider the size as a special constant symbol that appears in *every* type without mentioning it. Similarly, we consider the identity function $\mathbf{id}(x) = x$ as a function in any type t structure without mentioning it explicitly. We refer to the size of a structure s by $s.n$. In our notation, we usually do not distinguish between constant or function symbols and the actual values they take in a structure. As it is crucial for all of our considerations that the input numbers are linearly bounded in n, we pay special attention to the size of the values $s.f(i)$ and $s.C$ compared with $s.n$. If c is a natural number such that $s.f(i) \leq cs.n$ for each $i \in [n]$ and each $f \in \mathbf{t}$, and $s.C \leq cs.n$ for each $C \in \mathbf{t}$, then we say that s has *magnification bound* c or that s is *c-bounded* or $m(s) \leq c$. Intuitively, the magnification bound plays a similar role for RAM data structures as the size of the alphabet does for strings.

The restriction of the *size* of the numbers to be linearly bounded in n, while it may seem at first artificial, is an important part of our model. On one hand, we do not want arbitrary large values in our RAM registers, as, e.g., allowing linear increase in register length would result in exponential increase in the value represented. On the other hand, limiting the values strictly to n would result in at least technical awkwardness. For example, in the natural representation of graphs by adjacency lists, the numbers that occur are bounded by the number of vertices plus the number of edges.

Our convention does not prevent RAM algorithms from manipulating arbitrarily large words or integers at a higher level: e.g., as in real computers, where the word size is fixed, a large integer x can be represented in contiguous registers as a list of smaller integers $x_0, x_1, \ldots, x_k$, $x_i < b$, $x = x_0 + x_1 b + \cdots + x_k b^k$, i.e., in base b, and we take here $b = \theta(n)$.

For types $\mathbf{t}, \mathbf{t}'$ a $(\mathbf{t}, \mathbf{t}')$-*RAM function* Γ is a function which maps, for some constants $c(\Gamma), d(\Gamma)$, every RAM data structure of type $\mathbf{t}$ with magnification bound $c(\Gamma)$ to a RAM data structure of type $\mathbf{t}'$ with magnification bound $d(\Gamma)$. Γ is *linear* if $\Gamma(s).n = O(s.n)$.

A $\mathbf{t}$-*RAM decision problem* L is a set of RAM data structures of type $\mathbf{t}$ with magnification bound c for some constant c.

Our RAM data structures are minimally structured. They can encode more common data structures such as graphs, lists, lists of lists, etc. in a transparent way. We give two examples for such encodings that will be used in what follows.

Example 1. (a) Many data can be viewed at a lower level as mixed lists of nonnegative integers and special symbols, e.g., letters, parentheses, operators, equality signs, and so on, from a fixed alphabet Σ. For example, the system of equations

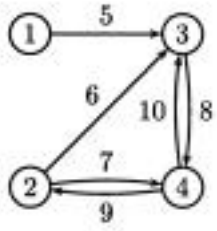
$$f_2(x) = 37 + f_0(x), \quad f_4(x) = f_1(x) - 51$$

can be represented by a list of 27 elements: $f, 2, (, x,), \ =, 37, \ldots, -, 51$. To distinguish numbers from other symbols, we represent each symbol of Σ by a number between 0 and $k-1$ and each number a by $a+k$, where k is the size of Σ. If, in the example above, the alphabet Σ has the 9 symbols $f, x, (,), \ = \ , \ + \ , \ - \ , \ \times \ , ;$, the system of equations can be represented by the RAM data structure s of type $\{g\}$, where $s.n = 27$ and $s.g(0) = 0$, $s.g(1) = 2 + 9 = 11$, $s.g(2) = 2, \ldots$, and $s.g(26) = 51 + 9 = 60$. Note that s has magnification bound 3.

(b) As a second example, we show how directed graphs can be represented by RAM data structures. The representation is an encoding of the adjacency list. Let G be a directed graph with vertex set V and edge set E. Let m and m' be the number of elements of V and E, respectively. We represent the vertices by the numbers $1, \ldots, m$ and the edges by the numbers $m+1, \ldots, m+m'$. If vertex 1 has out-degree l_1, then the numbers $m+1, \ldots \ .m+l_1$ represent the outgoing edges of vertex 1. The numbers $m+l_1+1, \ldots m+l_1+l_2$ represent the outgoing edges of vertex 2, and so forth.

The RAM data structure $s(G)$, which represents G, has type $\{m, m', f^-, f^+\}$, where m and m' are constant symbols and f^- and f^+ are function symbols. $s(G)$ is defined as follows.

- $s(G).n = m + m' + 1$.
- $s(G).m = m$.
- $s(G).m' = m'$.
- $s(G).f^-(i) = 0$ for all $i \in \{1, \ldots, m\}$.
- $s(G).f^+(i)$ is the first outgoing edge from vertex i for all $i \in \{1, \ldots, m\}$.



i	$f^-(i)$	$f^+(i)$
0	0	0
1	0	5
2	0	6
3	0	8
4	0	9
5	1	3
6	2	3
7	2	4
8	3	4
9	4	2
10	4	3

FIG. 2. *An example graph G and part of its representation.*

- For each number i which represents an outgoing edge of vertex j, we have $f^-(i) = j$, and $f^+(i)$ is the number of the target vertex of i.

As an illustration, consider the representation of the graph G, which is shown in Figure 2.

We have $s(G).n = 11$. The function values $s(G).f^-(i)$ and $s(G).f^+(i)$ for all $i \in [11]$ are given in Figure 2. Note that we do not make any assumption on the order in which the outgoing edges of a vertex are represented.

RAM data structures are very similar to finite structures as they appear in finite model theory [9]. A finite structure consists of a universe (the set of basic elements), constants, relations, and functions. There are the following main differences:

- We do not make use of relation symbols.
- All function symbols are *unary*.
- In a RAM data structure of size n, the "universe" is $[n]$. Unlike in finite structures, we allow that functions and constants take values outside the "universe"—but only in $[cn]$ for some constant c.

2.2. RAMs and linear time. In this section, we define precisely our model of computation and the complexity class **DLIN**.

A *simple* $\{+, -, \times\}$*-RAM* M is a RAM with two *accumulators* A, B, a *special register* N, and registers R_i for every $i \geq 0$. Its program is a sequence $I(1), \ldots, I(r)$ of instructions, each of which is of one of the following forms:

- $A := c$ for some constant $c \geq 0$,
- $A := A * B$, where $* \in \{+, -, \times\}$,
- $A := N$,
- $N := A$,
- $A := R_A$,
- $B := A$,
- $R_A := B$,
- IF $A = B$ THEN $I(i_0)$ ELSE $I(i_1)$,
- HALT.

The meaning of most of these instructions is straightforward. If the accumulator A contains a number l, then the execution of the instruction $A := R_A$ copies the contents of register R_l into A, whence the execution of $R_A := B$ copies the contents of B into R_l. We require the last instruction $I(r)$ of the program to be HALT. If before the execution of an instruction $A := A - B$ the content of A is smaller than that of B, then A is set to 0.

More generally, we define analogously simple Op-RAMs for any set Op of binary and/or unary operations.

Let $\mathbf{t}$ be a type that consists of the constant symbols $C_1, \ldots, C_l$ (besides n) and function symbols $f_1, \ldots, f_k$. We say that a RAM data structure of type $\mathbf{t}$ *corresponds to* register values of a RAM, or vice versa, if

- the special register N contains $s.n$,
- the registers $R_{(s.n)k}, \ldots, R_{(s.n)k+l-1}$ contain $s.C_1, \ldots, s.C_l$, and
- for each $i \in \{1, \ldots, k\}$, the consecutive registers $R_{(i-1)(s.n)}, \ldots, R_{i(s.n)-1}$ contain $s.f_i(0), \ldots, s.f_i(s.n-1)$, respectively.

At the beginning of a computation, on input s, the register values of the RAM correspond to s and A and B, and all registers R_i that are not determined by this correspondence have initial value 0.[2] The computation starts with instruction $I(1)$ and finishes when it executes a HALT statement. M computes the RAM function Γ if, for each $c(\Gamma)$-bounded input s, the computation terminates and, at the end of the computation, the register values correspond to $\Gamma(s)$.

Let **DLIN** denote the class of RAM functions Γ that can be computed in linear time $O(n)$ (i.e., using $O(n)$ instructions) on a simple $\{+, -, \times\}$-RAM which uses only numbers of value $O(n)$ for inputs of size n. We note that our convention that a RAM that works in time $O(n)$ should use only values $O(n)$ as register contents and addresses naturally implies that such a RAM also uses only a linear memory. This is a natural restriction as one expects that an algorithm which works in linear time $O(n)$ does not use space more than $O(n)$.

Although it is not decidable whether a given RAM M uses only linear values for RAM data structures with a given magnification bound c, by adding overflow test instructions after each instruction of M, every RAM M can be easily transformed into a RAM M' such that, for each $c > 0$, there is a $d > 0$ with the following two properties:

1. M' uses only values $< ds.n$ on c-bounded inputs s.
2. If M uses only values $< ds.n$ on c-bounded inputs s, then M' simulates the behavior of M.

We call such a RAM M' *safe* and will assume safety for all the RAMs that we consider in this article. In some of our proofs, we will make use of multimemory RAMs, i.e., RAMs with several sequences $R^1, \ldots, R^k$ of registers and instructions $A := R^j_A$ and $R^j_A := B$ for every $j \leq k$, with the obvious semantics. This does not change the power of RAMs significantly [17]. In particular, the class defined as **DLIN** does not change if one uses multimemory RAMs instead of RAMs. More generally, complexity classes on RAMs are robust. Many operations and different statements can be introduced without changing the class.

PROPOSITION 2.1. *A $\{+, -, \times, \div 2\}$-RAM[3] M working in time $O(n)$ and using only integers of value $O(n)$ can be simulated in time $O(n)$ on a $\{+\}$-RAM M', which again uses only integers of value $O(n)$.*

Proof. We are going to demonstrate how multiplication can be simulated on a multimemory $\{+\}$-RAM. The simulation of the other operations is similar and even simpler. More generally, it was proved in [17] that the proposition holds for any set of linear time turing computable (LTTC) operations, such as $-$, $\div 2$, shift, XOR, concatenation, etc., and for all time bounds $t(n) \geq n$ in place of $O(n)$.

[2] In our context, the initialization with 0 could also be done by the program itself, as only the first dn registers are used during the computation for some fixed d.

[3] Here $\div 2$ denotes the unary operation "division by 2."

Let c be a constant such that M uses, on inputs of size n, only numbers that are smaller than cn. The main idea is to use precomputed tables, in particular, a multiplication table for integers that are smaller than $b := \lceil \sqrt{cn} \rceil$. The simulation of M by M' proceeds in two stages.

Precomputation.

(1) Compute $b = \lceil \sqrt{cn} \rceil$.
(2) For each integer $i < cn$, compute and store in two tables $T_{\mathrm{mod}b}$ and $T_{\div b}$ the lower and the higher part of i, respectively. More precisely, let $T_{\mathrm{mod}b}[i] = i \bmod b$ and $T_{\div b}[i] = i \div b$.
(3) For each integer $i < b$, compute and store $T_{\times b}[i] = ib$.
(4) For every $i, j < b$, compute $T[ib + j] = ij$. T can be seen as a multiplication table for integers that are less than b.

Each of these tables can be computed by M' in time $O(s.n)$. As an example, the computation of T is based on the following induction:

$$T[ib + (j+1)] = \begin{cases} T[ib + j] + i & \text{if } j + 1 < b, \\ 0 & \text{otherwise.} \end{cases}$$

Main computation. M' works exactly as M does, except that each multiplication statement $A := A \times B$ is simulated in constant time by making use of the precomputed tables. Let x and y denote the numbers stored in A and B, respectively. As the product is not allowed to be as large as b^2, one of the factors, say, x, must be smaller than b. Hence

$$\begin{aligned} x \times y &= x \times (y \bmod b) + x \times b \times (y \div b) \\ &= T[T_{\times b}[x] + T_{\mathrm{mod}b}[y]] + T_{\times b}[T[T_{\times b}[x] + T_{\div b}[y]]]. \end{aligned}$$ □

2.3. Affine reductions. In order to study **DLIN**-complete problems and, more generally, to define a machine-independent reduction adapted for low complexity problems, we now introduce some simple algebraic notions and results about transformations of RAM data structures.

DEFINITION 2.2. *A* nondecreasing affine function*, for short* affine function*, is a function $A : \mathbb{N}^k \to \mathbb{N}$ of the form $A(x_1, \ldots, x_k) = a_0 + a_1 x_1 + \cdots + a_k x_k$, where the constants $a_1, \ldots, a_k$ are nonnegative integers and a_0 is any constant integer such that $a_0 \geq 0$ if $a_1 = a_2 = \cdots = a_k = 0$.*

DEFINITION 2.3. *Let T be a RAM function which maps RAM data structures of type $\mathbf{t}$ to RAM data structures of type $\mathbf{t}'$. T is an* affine transformation *if the following conditions hold.*

(1) *There are constants l, $d_1, \ldots, d_l$ and affine functions $\alpha_1, \ldots, \alpha_l$ such that, for each RAM data structure s of type $\mathbf{t}$, $T(s).n = \sum_{i=1}^{l} d_i \alpha_i(s.n)$.*
(2) *For each constant symbol $C' \neq n$ of $\mathbf{t}'$, there is an affine function $\beta_{C'}$ such that, for each RAM data structure s of type $\mathbf{t}$, $T(s).C' = \beta_{C'}(s.\overline{C})$, where $\overline{C}$ is the vector of all constant symbols of $\mathbf{t}$, including n.*
(3) *For each function symbol g of $\mathbf{t}'$, each $i \leq l$, and each $r < d_i$, there is a function symbol $f_{g,i,r}$ of $\mathbf{t}$, possibly the identity function symbol, and an affine function $A_{g,i,r}$ such that, for each $i \leq l$, $r < d_i$, for each s, and $x < \alpha_i(s.n)$,*

$$T(s).g\left(r + d_i x + \sum_{j<i} d_j \alpha_j(s.n)\right) = A_{g,i,r}(s.\overline{C}, s.f_{g,i,r}(x)),$$

where $\overline{C}$ is as above and $s.f_{g,i,r}(x)$ is interpreted as 0 whenever $x \geq s.n$.

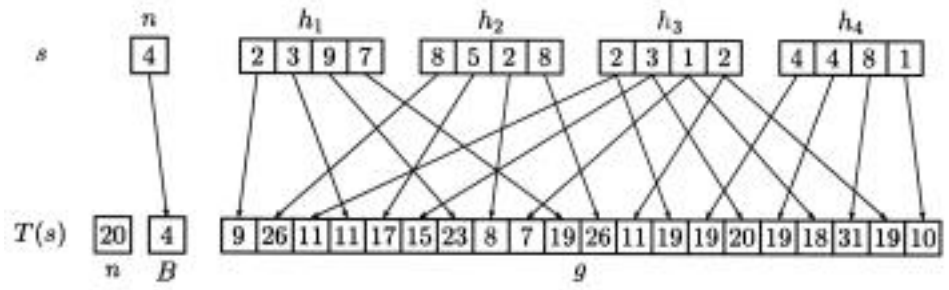


FIG. 3. *Example of an affine transformation.*

Note that each $T(s).g(y)$ depends on only one value $s.f_{g,i,r}(x)$. Also note that the size of the transformed input $T(s)$ is given by an affine function with respect to the size of s.

Intuitively, affine transformations allow linear transformations as well as "shuffle" and "concatenation" of the functions of the input structure. We illustrate the definition with a very simple example.

Example 2. Let $\mathbf{t}$ consist of the function symbols $h_1, \ldots, h_4$, and let $\mathbf{t}'$ consist of the function symbol g and the constant symbol B. We define an affine transformation from $\mathbf{t}$-structures to $\mathbf{t}'$-structures by choosing the following parameters.

- $l = 2$.
- $d_1 = 3$, $d_2 = 2$.
- $\alpha_1(n) = \alpha_2(n) = n$.
- $\beta_B(n) = n$.
- $f_{g,1,0} = h_1$, $f_{g,1,1} = h_2$, $f_{g,1,2} = h_3$.
- $f_{g,2,0} = h_3$, $f_{g,2,1} = h_4$.
- $A_{g,1,0}(n, y) = n + 2y + 1$.
- $A_{g,1,1}(n, y) = 3y + 2$.
- $A_{g,1,2}(n, y) = 4y + 3$.
- $A_{g,2,0}(n, y) = 3n + y + 5$.
- $A_{g,2,1}(n, y) = 3y + 7$.

Figure 3 shows s and $T(s)$ for an input structure s with $s.n = 4$. For example, $g(14) = 20$ because $14 = 0 + 2 \times \mathbf{1} + 3 \times 4$ and $A_{g,2,0}(4, f_{g,2,0}(\mathbf{1})) = 3 \times 4 + 3 + 5$. The function g in $T(s)$ can be seen as the concatenation of the shuffle of three linear transformations of h_1, h_2, h_3, respectively, and the shuffle of two linear transformations of h_3 and h_4, respectively.

Our transformations compose well as is shown by the result that follows. The proof is straightforward.

LEMMA 2.4. *If T and T' are affine transformations, then the composed transformation $T' \circ T$ is also affine.*

DEFINITION 2.5. *Let L, L' be two decision problems, i.e., sets of RAM data structures. An* affine reduction *from L to L' is an affine transformation T such that, for each RAM data structure s, we have $s \in L$ if and only if $T(s) \in L'$. If such a reduction exists, we write $L \leq_a L'$.*

Remark 1. The intention of the notion of affine reduction is two-fold. Affine reductions are

- *machine-independent* and
- at least as restrictive as the classical notions of reductions as indicated by the following proposition.

PROPOSITION 2.6. *An affine transformation can be computed by a deterministic Turing machine that works both in linear time and logarithmic space (when RAM data structures are represented by strings).*

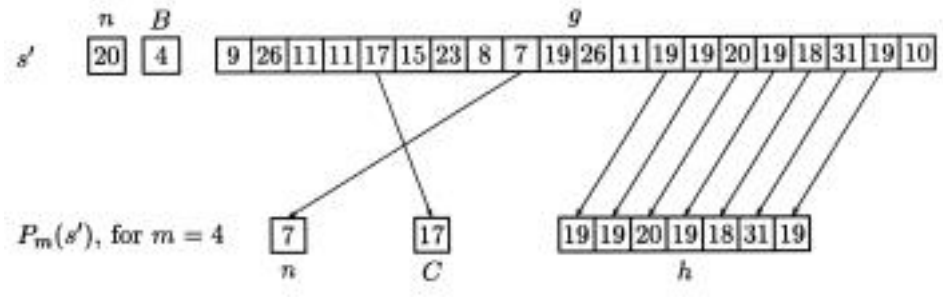


FIG. 4. *Example of an affine projection.*

Remark 2. In fact, affine reductions refine the very strict reset-log-lin reductions defined and used by [5].

In this article, we are mainly interested in RAM functions and not in decision problems. For a reduction between RAM functions, we need a means to extract a RAM data structure from another, possibly larger, RAM data structure. For this purpose, we define affine projections, which are much stricter than affine transformations.

DEFINITION 2.7. *Let* $\mathbf{t}$ *and* $\mathbf{t}'$ *be types of RAM data structures. An* affine projection P *is a transformation*

$$P : (m, s) \longrightarrow P_m(s),$$

where $m \in \mathbb{N}$ *and* s *and* $P_m(s)$ *are RAM data structures of respective types* $\mathbf{t}$ *and* $\mathbf{t}'$*, such that* $P_m(s)$ *is defined as follows.*

(1) *For each constant symbol* C *of* $\mathbf{t}'$*, and, in particular, for the symbol* n*, the values* $P_m(s).C$ *are determined in one of the following two ways.*
   - *Either there is a constant symbol* D_C *of* $\mathbf{t}$ *such that, for each* $\mathbf{t}$*-structure* s *and each* m*, it holds that* $P_m(s).C = s.D_C$*, or*
   - *there are a function symbol* f_C *of* $\mathbf{t}$ *and an affine function* $\alpha_C : \mathbb{N} \to \mathbb{N}$ *such that, for each* $\mathbf{t}$*-structure* s *and each* $m \in \mathbb{N}$*, it holds that* $P_m(s).C = s.f_C(\alpha_C(m))$*.*

(2) *For each function symbol* g *of* $\mathbf{t}'$*, there are a function symbol* f_g *of* $\mathbf{t}$ *and an affine function* $A_g : \mathbb{N}^2 \to \mathbb{N}$ *such that, for each* $\mathbf{t}$*-structure* s *and each* $m, a \in \mathbb{N}$*, it holds that* $P_m(s).g(a) = s.f_g(A_g(m, a))$*.*

In this definition, we again stick to the convention that $s.f(x) = 0$ whenever $x \geq s.n$.

Intuitively, the additional argument m in P corresponds to the input size, and each component of the output $P_m(s)$, e.g., $P_m(s).g(a) = s.f_g(A_g(m, a))$, is a copy of some component of the original output s, the address of which is given by some affine function.

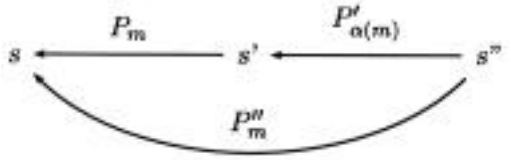
Example 3. We continue Example 2. Again let $\mathbf{t}'$ consist of the function symbol g and the constant symbol B, and let $\mathbf{t}$ have the constant symbol C and the function symbol h. In Figure 4, we illustrate $P_m(s')$ for $m = 4$ and the affine projection P, which is given by setting

- $f_n = g$ and $\alpha_n(m) = 2m$,
- $f_C = g$ and $\alpha_C(m) = m$, and
- $f_h = g$ and $A_h(m, a) = 3m + a$,

where $s' = T(s)$ is the transformed structure from Example 2. The reader should note how m is used to determine where the values for $P_m(s').n$, $P_m(s').C$, and $P_m(s').h$ can be found in $s'.g$.

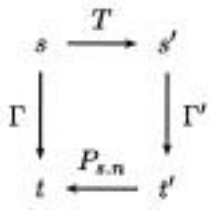
Affine projections compose well as is shown by the technical result that follows. The proof is straightforward.

LEMMA 2.8. *If P and P' are affine projections of appropriate types and $\alpha : \mathbb{N} \to \mathbb{N}$ is an affine function, then the mapping P'' defined by the commutative diagram*



i.e., such that $P''(m, s'') = P(m, P'(\alpha(m), s''))$, is also an affine projection.

DEFINITION 2.9. *Let Γ, Γ' be two RAM functions. An* affine reduction *ρ from Γ to Γ' is a pair (T, P), where T is an affine transformation and P is an affine projection such that the following diagram is commutative for all RAM structures s, s', t, t' of the appropriate types.*



In other words,

$$\Gamma(s) = P(s.n, \Gamma'(T(s)))$$

for each RAM data structure s.

If ρ is an affine reduction from Γ to Γ', then we write $\Gamma \to_\rho \Gamma'$. To express that there exists *an affine reduction from Γ to Γ', we write $\Gamma \leq_a \Gamma'$.*

Examples and properties of affine reductions. Affine reductions between RAM functions are a generalization of affine reductions between decision problems.

We note that the relation $\leq_a$ is reflexive and transitive.

- For each RAM function, we have $\Gamma \to_\rho \Gamma$ by the affine reduction $\rho = (T, P)$, where $T(s) = s$ and $P_m(t') = t'$.
- The transitivity follows from Lemmas 2.4 and 2.8.

Next, we are going to illustrate the notion of affine reductions by two more involved examples. First, we show that, for each $k > 3$, the classical reduction from k-satisfiability (-SAT) to 3-SAT can be turned into an affine reduction.

Example 4. Let $k \geq 3$ be fixed. The problem k-SAT is defined as follows.

Input. An integer p and n clauses $C_0, \ldots, C_{n-1}$ of k literals with variables from $v_0, \ldots, v_{p-1}$.

Question. Is $C_0 \wedge \cdots \wedge C_{n-1}$ satisfiable?

We represent an input for k-SAT by a RAM data structure s with one constant P and $2k$ functions $f_1, \ldots, f_k$ and $g_1, \ldots, g_k$. Here P represents the number of variables,

and, for each $i \leq s.n$ and each $j \leq k$, the jth literal of clause C_i is $v_{f_j(i)}$ if $g_j(i) = 1$ and $\neg v_{f_j(i)}$ if $g_j(i) = 0$.

The idea of the reduction is to transform a clause $(l_1^i \vee l_2^i \vee \cdots \vee l_k^i)$ into clauses $(\neg y_2^i \vee l_1^i \vee l_2^i)$, $(\neg y_3^i \vee y_2^i \vee l_3^i)$, $\ldots$, $(\neg y_{k-2}^i \vee y_{k-3}^i \vee l_{k-2}^i)$, and $(y_{k-2}^i \vee l_{k-1}^i \vee l_k^i)$, where the y_j^i, $i < n, 2 \leq j \leq k-2$, are new variables which intuitively represent the values of $l_1^i \vee \cdots \vee l_j^i$.

It is now easy to verify that this transformation is affine for the above representation of k-SAT instances. We encode the variables y_j^i by the single-indexed variables $v_{p+(k-2)i+j}$. The transformation T from k-SAT instances to 3-SAT instances can be defined as follows.

(1) $l = 1$, $d_1 = k-2$, and $T(s).n = (k-2)s.n$.
(2) $T(s).P = s.P + (k-1)s.n$.
(3)
- $T(s).f_1((k-2)x) = s.P + (k-2)x + 2$, $T(s).g_1((k-2)x) = 0$ (corresponding to the first literal $\neg y_2^i$ in the first clause that replaces a clause of s).
- More generally, $T(s).f_1(r+(k-2)x) = s.P+(k-2)x+(r+2)$, $T(s).g_1(r+(k-2)x) = 0$ for each r, $0 \leq r \leq k-4$.
- $T(s).f_1((k-3)+(k-2)x) = s.P + (k-2)x + (k-2)$, $T(s).g_1((k-3)+(k-2)x) = 1$ (corresponding to the first literal of the last clause).
- $T(s).f_2((k-2)x) = s.f_1(x)$, $T(s).g_2((k-2)x) = s.g_1(x)$ and $T(s).f_2((k-3)+(k-2)x) = s.f_{k-1}(x)$, $T(s).g_2((k-3)+(k-2)x) = s.g_{k-1}(x)$ (corresponding to the second literal of the first and last clause, respectively).
- $T(s).f_2(r+(k-2)x) = s.P+(k-2)x+(r+1)$, $T(s).g_2(r+(k-2)x) = 1$ for all r, $1 \leq r \leq k-4$.
- $T(s).f_3(r+(k-2)x) = s.f_{r+2}(x)$, $T(s).g_3(r+(k-2)x) = s.g_{r+2}(x)$ for all r, $0 \leq r \leq k-4$.
- $T(s).f_3((k-3)+(k-2)x) = s.f_k(x)$, $T(s).g_3((k-3)+(k-2)x) = s.g_k(x)$.

As a second example, we sketch why the problem CONTRACT is complete for the class **NLIN** under affine reductions. This problem was introduced in [18], where it was also shown to be complete for **NLIN** under **DTIME**(n)-reductions. For simplicity, we restrict ourselves in the following to RAM data structures with only one unary function f.

The problem CONTRACT is defined as follows.

Input. A set C of constants, a set V of variables, a set $\mathcal{F}$ of unary function symbols, and a set γ of triples (f, u, v), where $f \in \mathcal{F}$, $u, v \in C \cup V$.

Question. Is there a function VAL : $C \cup V \to C$ which is the identity on C such that the equations $f(\mathrm{VAL}(u)) = VAL(v)$, $(f, u, v) \in \gamma$, define partial functions for each $f \in \mathcal{F}$?

We think of the triples (f, u, v) of γ as equations $f(u) = v$.

The proof that CONTRACT is complete for **NLIN** under affine reductions makes use of the following normal form theorem of Olive [29].

THEOREM 2.10. *Every decision problem L in* **NLIN** *with one unary function symbol f is expressible by a formula of the form*

$$\exists \overline{f}\ \ \forall x \varphi(x, \overline{f}, f, succ, zero),$$

where φ is a quantifier-free formula of the form $\bigwedge_{i=1}^{k} \sigma_i(x) = \tau_i(x)$. Here $\overline{f}$ is a tuple of unary function symbols, succ and zero denote the successor function and the everywhere 0 function, respectively, and the σ_i and τ_i are terms over $f, \overline{f}, succ, zero$. More precisely, a 1-bounded structure s is in L if and only if $(s, succ, zero) \models \exists \overline{f}$

$\forall\, x\varphi(x)$, *where succ and zero are defined by* $succ(i) = i + 1$ *for all* $i < n - 1$, $succ(n - 1) = n - 1$, *and* $zero(i) = 0$ *for all* $i < n$.

From this theorem, it is easy to deduce the following corollary.

COROLLARY 2.11. *For any decision problem* L *in* **NLIN**, *it holds that* $L \leq_a$ *CONTRACT.*

We illustrate the proof of the corollary by the following example.

Example 5. Let L be expressible by the following sentence.

$$\exists g, h \quad \forall x f(g(x)) = h(x) \wedge h(\mathrm{succ}(x)) = \mathrm{zero}(g(x)).$$

First, for each structure s, $s \in L$ if and only if the following formula is satisfiable on the domain $[n]$.

$$(*) \qquad \mathrm{Diag}(\mathrm{succ}) \wedge \mathrm{Diag}(\mathrm{zero}) \wedge \mathrm{Diag}(f) \wedge \exists g, h \quad \forall x \varphi(x),$$

where

- $\varphi(x) \equiv f(g(x)) = h(x) \wedge h(\mathrm{succ}(x)) = \mathrm{zero}(g(x))$,
- $\mathrm{Diag}(\mathrm{succ}) \equiv (\bigwedge_{i<n-1} \mathrm{succ}(i) = i + 1) \wedge (\mathrm{succ}(n - 1) = n - 1)$,
- $\mathrm{Diag}(\mathrm{zero}) \equiv \bigwedge_{i<n} \mathrm{zero}(i) = 0$, and
- $\mathrm{Diag}(f) \equiv \bigwedge_{i<n} f(i) = s.f(i)$.

Second, we unfold the formula $\exists g, h\, \forall\, x\varphi(x)$, and hence we replace it by a formula $\bigwedge_{i<n} \varphi'(i)$. More precisely, let $(C, V, \mathcal{F}, \gamma)$ be the instance of CONTRACT which is given by $C = \{0, \ldots, n - 1\}$, $V = v_0, \ldots, v_{3n-1}$, $\mathcal{F} = \{f, g, h, \mathrm{succ}, \mathrm{zero}\}$, and the set γ is defined below. Intuitively, the variables $v_0, \ldots, v_{n-1}$ correspond to the terms $g(x)$, the variables $v_n, \ldots, v_{2n-1}$ correspond to the terms $h(x)$, and the variables $v_{2n}, \ldots, v_{3n-1}$ correspond to the terms $\mathrm{succ}(x)$.

The equations of γ are those which are given by Diag(succ), Diag(zero), and Diag(f) plus the following equations for every $i < n$:

- $g(i) = v_i$,
- $h(i) = v_{i+n}$,
- $f(v_i) = v_{i+n}$,
- $\mathrm{succ}(i) = v_{i+2n}$,
- $h(v_{i+2n}) = 0$.

Clearly, the above formula $(*)$ is satisfiable on $[n]$ if and only if γ is satisfiable on $C = [n]$. Hence $s \in L$ if and only if $(C, V, \mathcal{F}, \gamma) \in$ CONTRACT.

Moreover, this reduction from L to CONTRACT can be made affine. To this end, we represent instances $(C, V, \mathcal{F}, \gamma)$ by RAM data structures with constant symbols C, V, F and functions f_1, f_2, f_3. The symbols $s.C$, $s.V$, and $s.F$ represent the size of C, V, and $\mathcal{F}$, respectively, in a natural way. The value $s.n$ coincides with the number of triples in γ. The ith triple of γ is represented by $(s.f_1(i), s.f_2(i), s.f_3(i))$, where we view the elements of C to be numbered from 0 to $s.C - 1$, the elements of V from $s.C$ to $s.C + s.V - 1$, and the elements of $\mathcal{F}$ from 0 to $s.F - 1$.

It is straightforward to see that the above reduction can be made affine in this framework. For example, if the successor function symbol is numbered f_0 and we represent the successor function at the beginning of each instance s, then we get, for each $i < s.n - 1$, $s.f_1(i) = 0, s.f_2(i) = i, s.f_3(i) = i + 1$ (corresponding to the triple $(\mathrm{succ}, i, i + 1)$), and $s.f_1(s.n - 1) = 0, s.f_2(s.n - 1) = s.n - 1, s.f_3(s.n - 1) = s.n - 1$ (for $(\mathrm{succ}, s.n - 1, s.n - 1)$); in other words, we use the affine functions $\alpha_1(n) = n - 1$, $\alpha_2(n) = 1$ with the corresponding integers $d_1 = d_2 = 1$. For example, for $s.f_3$, we get the following values: $s.f_3(i) = i + 1$ for $i < \alpha_1(s.n) = s.n - 1$ and $s.f_3(i + \alpha_1(s.n)) = s.n - 1$ for $i < \alpha_2(s.n) = 1$ and hence for $i = 0$.

3. The algebraic framework. In this section, we are going to give two algebraic characterizations of **DLIN**. In subsection 3.1, we define simultaneous recursion schemes. These are recursion schemes that define several functions at once. From the structure containing these functions, the output data structure is extracted by an affine projection. In subsection 3.2, we introduce nonsimultaneous recursion schemes. These schemes allow more liberal operators, but they can define only a single function.

3.1. A linear simultaneous recursion scheme (LSRS). First, we define simultaneous recursion schemes. We allow two kinds of equations: standard equations that use arithmetical operations and a new kind that use two new operators, *Bounded Application* and *Equal-Predecessor*. Let us consider a function $f : [n] \to \mathbb{N}$. We define

Bounded Application.

$$f[x]_y := \begin{cases} f(x) & \text{if } x < y, \\ x & \text{otherwise.} \end{cases}$$

Equal-Predecessor.

$$f^{\leftarrow}(x) := \begin{cases} \max\{y < x : f(y) = f(x)\} & \text{if such a } y \text{ exists,} \\ x & \text{otherwise.} \end{cases}$$

It will turn out that the combination of these two operators captures the essence of random access read and write operations.

DEFINITION 3.1. *Let F be a set of unary function symbols, and let $f_1, \dots, f_k$ be unary function symbols that do not occur in F. For every $i \in \{0, \dots, k\}$, we write F_i for the set $F \cup \{f_1, \dots, f_i\}$. A* linear simultaneous recursion scheme (LSRS) *S for $f_1, \dots, f_k$ with respect to F is a sequence $E_1, \dots, E_k$ of equations, where each E_i is of one of the following two forms:*

- *(operation) $f_i(x) = g(x) * g'(x)$, where $g, g' \in F_{i-1}$ and $* \in \{+, -\}$,*
- *(recursion) $f_i(x) = g'[g^{\leftarrow}(x)]_x$, where $g \in F_{i-1}$ and $g' \in F_k$.*

Remark 3. We could choose other, more general, types of equations in the definition of LSRS, e.g., the following types:

- any operations that are computable in linear time on a Turing machine (see [17]) and also multiplication, which is not known to be computable in linear time;
- (bounded composition) $f_i(x) = g'[g(x)]_x$, where $g \in F_{i-1}$ and $g' \in F_k$;
- (bounded search)

$$f_i(x) = \begin{cases} \max\{y < x \mid g'(y) = g(x)\} & \text{if such a } y \text{ exists,} \\ x & \text{otherwise,} \end{cases}$$

  where $g \in F_{i-1}$ and $g' \in F_k$. (Note that the Equal-Predecessor is a bounded search with $g = g'$.)

It is a consequence of the results of this article that all of these schemes define the same class of functions, namely, **DLIN**. We have chosen the above definition mainly for two reasons:

- it is small in size, as it has only two types of equations;
- it will enable us to prove other characterizations of **DLIN** in a smooth way, e.g., a new recursion scheme without *simultaneous* recursion (see the linear recursion scheme (LRS) in 3.2).

LEMMA 3.2. *Let S be an LSRS for function symbols $f_1, \dots, f_k$ with respect to a set $F = \{f'_1, \dots, f'_l\}$ of function symbols. Then, for all integers m and functions*

$f'_1, \ldots, f'_l : [m] \to [m]$, there are unique functions $f_1, \ldots, f_k : [m] \to [m]$ which fulfill the equations of S with the overflow convention *that $f_i(a) = 0$ if the term defining $f_i(a)$ by an addition evaluates to a value $\geq m$.*

Proof. It is sufficient to observe that the computation of every term $f_i(x)$, $1 \leq i \leq k$, $x < m$, requires only the values of terms of the form $f_j(y)$, where the pair (y, j) precedes the pair (x, i) in the natural lexicographical order on pairs of natural numbers; so all terms can be computed in that order. □

Given a type $\mathbf{t}$, we write in the following $\mathbf{F_t}$ to denote the set $\{\mathbf{1}, \mathbf{n}, \mathbf{id}\} \cup \{\mathbf{f}_C \mid C \in \mathbf{t}\} \cup \{f \mid f \in \mathbf{t}\}$ of function symbols.

DEFINITION 3.3. *Let $\mathbf{t}_1, \mathbf{t}_2$ be types. Let Γ be a RAM function, which maps RAM data structures of type $\mathbf{t}_1$ to RAM data structures of type $\mathbf{t}_2$. Let S be an LSRS for $f_1, \ldots, f_k$ with respect to $\mathbf{F}_{\mathbf{t}_1}$. Γ is* linearly represented by *S if there are an integer c and an affine projection P such that, for each $c(\Gamma)$-bounded RAM data structure s, S defines functions $f_1, \ldots, f_k : [cs.n] \to [cs.n]$ with respect to the following functions $[cs.n] \to [cs.n]$:*

- *for every function symbol $f \in \mathbf{t}_1$,*

$$\mathbf{f}(i) = \begin{cases} s.f(i) & \text{if } i < s.n, \\ 0 & \text{else,} \end{cases}$$

- *$\mathbf{f}_C(i) = s.C$ for every constant symbol $C \in \mathbf{t}_1$,*
- *$\mathbf{1}(i) = 1$ for every i,*
- *$\mathbf{n}(i) = s.n$ for every i,*
- *$\mathbf{id}(i) = i$ for every i,*

and $P_{s.n}(s') = \Gamma(s)$, where s' is the RAM data structure of size $cs.n$ with the functions $f_1, \ldots, f_k$.

If a RAM function Γ is linearly represented by an LSRS, then we call Γ LSRS-definable*. We say that a RAM decision problem L of type $\mathbf{t}$ with magnification bound c is* LSRS-definable *if the following RAM function Γ, which maps RAM data structures of type $\mathbf{t}$ to 1-bounded RAM data structures of type $\{C\}$, is LSRS-definable. For each s, Γ is defined by*

- *$\Gamma(s).n = s.n$ and*
- $$\Gamma(s).C = \begin{cases} 1 & \text{if } s \in L, \\ 0 & \text{if } s \notin L. \end{cases}$$

Example 6. As an example, we consider an LSRS S for the RAM function Γ, which maps RAM data structures of type $\{f\}$ to RAM data structures of type $\{C\}$ such that, for each s, $\Gamma(s).n = s.n$ and $\Gamma(s).C = \max\{s.f(j) \mid j < s.n\}$.

S defines functions f_1, f_2, f_3, f_4 by the following equations:

$$\begin{aligned} f_1(x) &= f_3[\mathbf{1}^{\leftarrow}(x)]_x, \\ f_2(x) &= \mathbf{f}(x) - f_1(x), \\ f_3(x) &= f_1(x) + f_2(x), \\ f_4(x) &= \mathbf{n}(x) - \mathbf{1}(x). \end{aligned}$$

Note that $f_3[\mathbf{1}^{\leftarrow}(0)]_0 = 0$, by definition of Bounded Application and Equal-Predecessor, and that $f_3[\mathbf{1}^{\leftarrow}(i)]_i = f_3(i-1)$ for $i > 0$. S defines functions f_1, f_2, f_3 such that

- $f_1(i) = f_3(i-1)$,
- $f_2(i) = s.f(i) - f_2(i-1)$, and
- $f_3(i) = \max\{s.f(j) \mid j \leq i\}$ for each $i < s.n$.

To see this, recall, by our convention, that $b-a = 0$ if $b < a$, and, therefore, $a+(b-a) = \max\{a, b\}$ for all a, b. We get

$$\begin{aligned} f_3(i) &= f_1(i) + (s.f(i) - f_1(i)) \\ &= \max(f_1(i), s.f(i)) \\ &= \begin{cases} s.f(0) & \text{if } i = 0, \\ \max(f_3(i-1), s.f(i)) & \text{otherwise.} \end{cases} \end{aligned}$$

In particular, $f_3(s.n-1) = \max\{s.f(j) \mid j < s.n\} = \Gamma(s).C$. Hence S defines Γ with $c = 1$ and the affine projection P, which is given by $f_n = f_4$, $\alpha_n(m) = 0$, $f_C = f_3$, and $\alpha_C(m) = m - 1$.

It will be convenient to allow somewhat more relaxed recursion schemes. In particular, we allow, in the equation for any f'_i, instead of expressions $g(x) * g'(x)$, arbitrary arithmetical expressions using $-, +$, and functions $g \in F_{i-1}$, and we allow the use of *definition by cases*.

DEFINITION 3.4. *A function f_i is defined by cases if it is defined as*

$$f_i(x) = \begin{cases} g_1(x) & \text{if } g(x) \le h(x), \\ g_2(x) & \text{otherwise,} \end{cases}$$

where g, h, g_1, g_2 have to be in F_{i-1}.

LEMMA 3.5. *Any RAM function that is linearly represented by an LSRS using arbitrary arithmetic expressions and definition by cases can be linearly represented by an LSRS without definition by cases.*

Proof. A definition of a function f_i which uses a more complicated arithmetic expression can easily be replaced by a series of definitions of functions that use only terms of the form $g(x) * g'(x)$. Now we describe how definition by cases can be simulated. First, note that the convention that $a - b$ equals 0 whenever $a < b$ also implies, for all $a, b \in \mathbb{N}$, the following two statements.

- $a - (a - b) = \min(a, b)$,
- $1 - (a - b) = \begin{cases} 1 & \text{if } a \le b, \\ 0 & \text{otherwise.} \end{cases}$

Furthermore, if we restrict ourselves to the domain $[m]$ for some $m > 0$, it follows from our overflow convention for addition (cf. Lemma 3.2) that, for all $a, b, c \in [m]$,

$$\min(a, (m-1) + (1-c)) + \min(b, (m-1) + c) = \begin{cases} a & \text{if } c > 0, \\ b & \text{if } c = 0. \end{cases}$$

This allows us to simulate definitions by cases by using several new functions that are defined by using arithmetic operations ($+$ and $-$) only. □

Remark 4. It should be pointed out that the proof of Lemma 3.5 relies on the choice of the overflow convention. Although we view this choice as natural, we would like to mention other possibilities.

1. We could allow multiplication in LSRS. The simulation of definitions by cases could then simply be obtained by using

$$a \times c + b \times (1 - c) \begin{cases} a & \text{if } c = 1, \\ b & \text{if } c = 0. \end{cases}$$

It follows from Proposition 2.1 and Theorem 4.1 that this change would not increase the expressive power of LSRS.

2. Alternatively, definitions by cases could be simulated by allowing only multiplication with constants via

$$\min(a,(m-1)\times c)+\min(b,(m-1)\times(1-c))=\begin{cases}a & \text{if } c>0,\\ b & \text{if } c=0.\end{cases}$$

3. A further possibility would be to make use of *bounded composition* (cf. Remark 3) via

$$\min(a,g(c))+\min(b,g(1-c))=\begin{cases}a & \text{if } c>0,\\ b & \text{if } c=0,\end{cases}$$

where g is the easily definable function with $g(0)=0$ and $g(i)=m-1$ for all other i.

The next easy lemma states that the set of LSRS-definable RAM functions is closed under composition.

LEMMA 3.6. *Let Γ_1 be a RAM function which maps RAM data structures of type $\mathbf{t}_1$ to RAM data structures of type $\mathbf{t}_2$, and let Γ_2 be a RAM function which maps RAM data structures of type $\mathbf{t}_2$ to RAM data structures of type $\mathbf{t}_3$. If Γ_1 is LSRS-definable and Γ_2 is LSRS-definable, then $\Gamma_2\circ\Gamma_1$ is also LSRS-definable.*

It should be noted that the LSRS defined in this article is different from the LSRS of [34]. On one hand, it uses a smaller set of operations, on the other hand, the underlying domain of the defined functions is allowed to be linear in the size n of the input whereas that domain was exactly $[n]$ in [34].

3.2. A one-function one-equation linear recursion scheme. Is it possible to avoid simultaneous recursion in an LSRS, as asked in [34]? In this subsection, we give a positive answer to this question: each problem that is linearly representable by an LSRS can also be represented by a one-function one-equation recursion scheme, which we call LRS. The price to be paid is a slightly greater number of constructs for subterms of the LRS.

First, we define inductively what a *recursion term* $\sigma(x)$ with respect to a function symbol g and a type $\mathbf{t}$ is.

- 1, n, and x are recursion terms.
- C is a recursion term for each constant symbol C from $\mathbf{t}$.
- (Equal-Predecessor) $g^{\leftarrow}(x-\delta)$ is a recursion term for every integer $\delta>0$.
- (Bounded Application) If $\sigma(x)$ is a recursion term, then $g[\sigma(x)]_x$ is a recursion term.
- (Input application) If $\sigma(x)$ is a recursion term and f is a function symbol of $\mathbf{t}$, then $f(\sigma(x))$ is a recursion term.
- (Arithmetic operation) If $\sigma(x)$ and $\tau(x)$ are recursion terms, then, for each $*\in\{+,-\}$, $(\sigma(x)*\tau(x))$ is a recursion term.

Unnecessary brackets can be omitted.

DEFINITION 3.7. *A* linear recursion scheme (LRS) *with respect to g and $\mathbf{t}$ is an equation of the form $g(x)=\sigma(x)$ for some recursion term $\sigma(x)$ with respect to g and $\mathbf{t}$.*

Example 7.

$$g(x)=(g[g^{\leftarrow}(x-2)]_x+x)-f(x+C)$$

is an LRS with respect to g and the type $\mathbf{t}=\{f,C\}$.

LEMMA 3.8. *Let $\sigma(x)$ be a recursion term with respect to g and $\mathbf{t}$. Then, for all integers $m > 0$ and all structures s of type $\mathbf{t}$, there is a unique function $g : [m] \to [m]$, which satisfies the equation $g(a) = \sigma(a)$ for every $a < m$ with the following conventions:*

- *$\tau(a) = 0$ if, for a subterm τ of σ, $\tau(a)$ evaluates to a value which is not in $[m]$;*
- *$f(j) = s.f(j)$ for $j < s.n$, and $f(j) = 0$ for $j \geq s.n$, for every function symbol f of $\mathbf{t}$;*
- *$C = s.C$ for every constant symbol C of $\mathbf{t}$.*

We say that the function $g : [m] \to [m]$ is *defined by the LRS $g(x) = \sigma(x)$ in s*. We call $[m]$ the *range* of this definition.

Proof. For the proof, it is sufficient to verify that, for every a, the evaluation of $\sigma(a)$ needs only values $g(b)$ for numbers $b < a$. □

Remark 5. Note that $g(a) = \sigma(a)$ should also hold for $a = 0$; in this case, notice the interpretation of the possible subterms of the respective forms $g[\sigma(x)]_x$ and $g^{\leftarrow}(x - \delta)$ of respective values $\sigma(0)$ and $g^{\leftarrow}(0) = 0$.

DEFINITION 3.9. *Let E be an LRS given by $g(x) = \sigma(x)$. Let Γ be a RAM function. Γ is* linearly represented *by E if there is a constant c and an affine projection P, such that, for each $c(\Gamma)$-bounded RAM data structure s, E defines a function $g : [cs.n] \to [cs.n]$ in s, such that $P_{s.n}(s') = \Gamma(s)$, where s' is the RAM data structure of size $cs.n$ with the single function g.*

As in the case of LSRS, we can make use of definition by cases without changing the expressive power of LRS. The following lemma shows that simultaneous recursion schemes are no more powerful than nonsimultaneous schemes.

LEMMA 3.10. *Every RAM function that is linearly representable by an LSRS is also linearly representable by an LRS.*

Proof. Let Γ be a RAM function that is linearly represented by an LSRS S, which defines functions[4] $f_0, \ldots, f_{k-1}$ with respect to $\mathbf{F}_{\mathbf{t}_1}$, as in Definition 3.3, and let P be the corresponding affine projection. Let s be a $c(\Gamma)$-bounded RAM data structure of type $\mathbf{t}_1$. We write s' for the structure that is defined by S on s and s'' for $\Gamma(s) = P_{s.n}(s')$. To simplify notation in the following, we write n to denote $s.n$. Roughly, the idea is to first encode the functions $f_0, \ldots, f_{k-1} : [cn] \to [cn]$ by one function g. To ensure that the Equal-Predecessor operation works properly, the encodings of the different functions should have disjoint domain and disjoint range. This is done by using, for each $i < k$, only values that are congruent i modulo k to encode function f_i. As a tool for encoding and decoding operations, we additionally encode the function $(\div k)$ into g.

We are going to extend the domain of g in order to encode $\Gamma(s).f$, for each function symbol f of the output data structure, by function values of a contiguous interval.

More precisely, g will be defined on $[2kcn]$ such that

- for each $b < kcn$, it holds that $g(b) = b \div k$, and
- for every a and i, where $a < cn$ and $0 \leq i \leq k - 1$, it holds that

$$(*) \qquad g(kcn + ka + i) = kcn + kf_i(a) + i.$$

In other terms, for $b \geq kcn$ and $b \bmod k = j$, the value of $g(b)$ is $kcn + kf_j((b - kcn) \div k) + j$. We are going to define the value $g(b)$ for $b < 2kcn$ by case distinction on $b \bmod k$, as described in (1)–(3) below.

[4] We enumerate the functions from 0 to $k - 1$ to facilitate computations modulo k.

(1) Let the function symbols of $\mathbf{F}_{\mathbf{t}_1} = \{f \mid f \text{ function in } \mathbf{t}_1\} \cup \{f_C \mid C \in \mathbf{t}_1\} \cup \{\mathbf{1}, \mathbf{n}, \mathbf{id}\}$ be enumerated by $\{f^0, \ldots, f^{l-1}\}$. For the sake of uniformity, we modify S to limit the occurrence of these function symbols. First, the old function symbols $f_0, \ldots, f_{k-1}$ and all of their occurrences in S are replaced by $f_l, \ldots, f_{k+l-1}$, respectively. Next, we introduce l new functions $f_0, \ldots, f_{l-1}$ and l equations which define them. If f^i is from $\mathbf{t}_1$, the defining equation is simply $f_i(x) = f^i(x)$. If f^i is $\mathbf{id}$, then the equation is $f_i(x) = x$; if it is $\mathbf{1}, \mathbf{n}$, or f_C, then the equation is $f_i(x) = 1$, $f_i(x) = n$, or $f_i(x) = C$, respectively. We refer to these equations as *input equations*. Finally, we replace in S all occurrences of functions from $\mathbf{F}_{\mathbf{t}_1}$ by the respective function symbols f_i.
It is straightforward that the modified system with the above $k+l$ equations is equivalent to S, under the obvious semantics, although S is no longer an LSRS. From now on, we refer to the new system by S, and k denotes the number of equations of S. After these replacements, S no longer refers to any functions of $\mathbf{F}_{\mathbf{t}_1}$ except in the input equations.

(2) The equations of S are combined into one equation $g(y) = \sigma(y)$ as follows.[5] The term $\sigma(y)$ consists of a definition by cases, depending on the size of y and on $y \bmod k$. More precisely, we define a recursion term $\sigma_i(y)$ corresponding to f_i for every $i < k$ and define g by

$$g(y) = \begin{cases} 0 & \text{if } y \leq k-1, \\ g(y-k)+1 & \text{if } k \leq y < kcn, \\ \sigma_i(y) & \text{if } y \geq kcn \text{ and } y \bmod k = i. \end{cases}$$

Of course, the two first cases imply that $g(y) = y \div k$ for each $y < kcn$, as required. This allows us to express $y \bmod k$ (for $kcn \leq y < 2kcn$) as $(y - kcn) - kg(y - kcn) = (y - kcn) - \sum_{j=1}^{k} g(y - kcn)$.
The case distinction itself is done along the same lines as in the proof of Lemma 3.5 above.
Now we describe the construction of the recursion terms $\sigma_i(y)$.

- If E_i is an input equation, then $\sigma_i(y)$ is constructed as follows.
  - If the right-hand side is a constant C (possibly 1 or n), then $\sigma_i(y)$ is defined as $kcn + kC + i$.
  - If the right-hand side is x, then $\sigma_i(y)$ is defined as $kcn + kg(y - kcn) + i$.
  - If the right-hand side is $f^i(x)$, then $\sigma_i(y)$ is defined as $kcn + kf^i(g(y - kcn)) + i$.

  We justify why this definition of $\sigma_i(y)$ is correct. We do this for the third case; the other two cases are even simpler. Let $b = kcn + ka + i$ with $a < cn$. Then $g(b - kcn) = g(ka + i) = (ka + i) \div k = a$, and $\sigma_i(b) = kcn + kf^i(a) + i = kcn + kf_i(a) + i$, as required.
- If E_i is of the form $f_i(x) = f_j(x) - f_{j'}(x)$, then $\sigma_i(y)$ is defined by

$$\sigma_i(y) = (g(y - \delta) - kcn - j) - (g(y - \delta') - kcn - j') + kcn + i,$$

  where $\delta = i - j$, $\delta' = i - j'$, and, by definition of LSRS, $i > j, j'$. To verify the correctness of this expression, let $a \in [cn]$, $b = kcn + ka + i$, where $i < k$. If $g(b - \delta) = g(kcn + ka + j)$ equals $kcn + kf_j(a) + j$ and

[5] We choose y instead of x to distinguish the variable of the LRS from the variable of the LSRS.

$g(b-\delta') = g(kcn+ka+j')$ equals $kcn + kf_{j'}(a) + j'$, then

$$\begin{aligned}(g(b-\delta) - kcn - j) - (g(b-\delta') - kcn - j') + kcn + i \\ = (kf_j(a) - kf_{j'}(a)) + kcn + i \\ = k(f_j(a) - f_{j'}(a)) + kcn + i,\end{aligned}$$

as required.

- The case in which E_i is of the form $f_i(x) = f_j(x) + f_{j'}(x)$ is similar to the previous subtraction case. However, it requires an additional case distinction because of our convention that, in universe $[cn]$, the result of an addition is 0 in case of overflow, which is not the case in the enlarged universe $[2kcn]$ if a sum lies between kcn and $2kcn$. More precisely, $\sigma_i(y)$ is defined by

$$\sigma_i(y) = \begin{cases} \tau(y) + kcn + i & \text{if } \tau(y) < kcn, \\ kcn + i & \text{otherwise,} \end{cases}$$

where $\tau(y) = (g(y-\delta) - kcn - j) + (g(y-\delta') - kcn - j')$ with $\delta = i - j$ and $\delta' = i - j'$. The proof of the correctness of $\sigma_i(y)$ is similar to the previous subtraction case.

- If E_i is of the form $f_i(x) = f_{j'}[{f_j}^{\leftarrow}(x)]_x$, where $j < i$ and we assume without loss of generality that[6] $i \le j'$, then $\sigma_i(y)$ is defined by

$$\sigma_i(y) = g[g^{\leftarrow}(y-\delta) + \delta']_y - j' + i,$$

where again $\delta = i - j$ and $\delta' = j' - j$.

To justify this replacement, we have to ensure that the encoding of several functions into one function does not produce any side-effects when the Equal-Predecessor operator is applied. It is crucial here that, for each $i < k$, those function values of g which encode function values of f_i are congruent to i modulo k. To be more precise, let $b = kcn + ka + i$, with $a < cn$. Then $b - \delta = kcn + ka + i - (i - j) = kcn + ka + j$. Only two cases may arise.

(a) ${f_j}^{\leftarrow}(a) = a$. This means that, for no $a' < a$ does it hold that $f_j(a') = f_j(a)$. Hence there is no $a' < a$ such that $g(kcn + ka' + j) = g(kcn + ka' + j)$. The definition of g ensures that, for each $e \ge kcn$, $g(e) \bmod k = j$ if and only if $e \bmod k = j$ and that, for all $e, e' \in [2kcn]$, if $g(e) = g(e')$, either $e, e' < kcn$ or $e, e' \ge kcn$. Hence $g^{\leftarrow}(kcn+ka+j) = kcn+ka+j$, and $g^{\leftarrow}(b-\delta)+\delta' = kcn+ka+j' \ge kcn + ka + i = b$. Hence

$$\begin{aligned}\sigma_i(b) &= g[g^{\leftarrow}(b-\delta) + \delta']_b - j' + i \\ &= (kcn + ka + j') - j' + i \\ &= kcn + ka + i \\ &= kcn + kf_{j'}[{f_j}^{\leftarrow}(a)]_a + i \\ &= kcn + kf_i(a) + i \\ &= g(b),\end{aligned}$$

as required.

[6] If $i > j'$, then we first may add a new function f_l into S, with $l > i$, defined by the new equation $f_l(x) = f'_{j'}(x)$ and replace E_i by $f_i(x) = f_l[{f_j}^{\leftarrow}(x)]_x$.

(b) ${f_j}^{\leftarrow}(a) = a'$ for some $a' < a$. Then $g^{\leftarrow}(kcn+ka+j) = kcn+ka'+j$. In consequence,

$$g^{\leftarrow}(b-\delta)+\delta' = kcn+ka'+j' < kcn+ka+i = b.$$

Hence

$$\begin{aligned}\sigma_i(b) &= g[g^{\leftarrow}(b-\delta)+\delta']_b - j' + i\\ &= g(kcn+ka'+j') - j' + i\\ &= (kcn + kf_{j'}(a') + j') - j' + i\\ &= kcn + kf_{j'}(a') + i\\ &= kcn + kf_{j'}[{f_j}^{\leftarrow}(a)]_a + i\\ &= kcn + kf_i(a) + i\\ &= g(b),\end{aligned}$$

as required.

(3) Now we complete the LRS for g. For simplicity, we assume that $\mathbf{t}_2$ consists of the constant symbol n and only one function symbol h. Let $j < k$ and α be an affine function such that, for all structures s, it holds that $\Gamma(s).n = P_{s.n}(s').n = s'.f_j(\alpha(s.n))$, and let $i < k$ and A be an affine function such that, for all structures s and all $a < \Gamma(s).n$, it holds that $\Gamma(s).h(a) = P_{s.n}(s').h(a) = s'.f_i(A(s.n,a))$.
We have constructed g in such a way that all function values $f_i(a)$ are somehow available in g, but we have to deal with two problems. First, the $f_i(a)$ appear only in an encoded way; second, they do not form a contiguous interval but are scattered (modulo k). Hence, before we can extract the values by an appropriate affine projection, we have to decode the function values and bring them together into one interval. To accomplish this, we enlarge the domain of g to $[(2k+2)cn]$ and complete the definition of g as specified below.

$$g(y) = \begin{cases} \text{as before} & \text{if } y < 2kcn,\\ g[g[k(y-2kcn)+kcn+i]_y - kcn]_y & \text{if } 2kcn \le y < (2k+1)cn,\\ g[g[k(y-(2k+1)cn)+kcn+j]_y - kcn]_y & \text{if } (2k+1)cn \le y < (2k+2)cn.\end{cases}$$

It follows from equation $(*)$, the definition of g on $[kcn]$, and this definition that, for all $a < cn$, it holds that

$$\begin{aligned}g(2kcn+a) &= g[g[k(2kcn+a-2kcn)+kcn+i]_{2kcn+a} - kcn]_{2kcn+a}\\ &= g[g[ka+kcn+i]_{2kcn+a} - kcn]_{2kcn+a}\\ &= g[g(ka+kcn+i) - kcn]_{2kcn+a}\\ &= g[kcn + kf_i(a) + i - kcn]_{2kcn+a}\\ &= g[kf_i(a)+i]_{2kcn+a}\\ &= f_i(a).\end{aligned}$$

Analogously, we get that, for all $a < cn$, $g((2k+1)cn+a) = f_j(a)$. Now it is easy to define an affine projection P' which extracts $\Gamma(s)$ from the structure

s''' with the function g and the constant $s.n$ via $\Gamma(s).n = s'''.g(\alpha(s.n)+(2k+1)cs.n)$ and

$$\begin{aligned}\Gamma(s).h(a) &= P_{s.n}(s').h(a)\\ &= s'.f_i(A(s.n,a))\\ &= s'''.g(2kcs.n + A(s.n,a)).\end{aligned}$$ □

4. LRSs and linear time. In this section, we show that LSRS and LRS, which were defined in the previous section, exactly characterize the complexity class **DLIN**.

THEOREM 4.1. *Let Γ be a RAM function. Then the following assertions are equivalent.*

1. Γ *belongs to* **DLIN**.
2. Γ *is linearly represented by some LSRS.*
3. Γ *is linearly represented by some LRS.*

Since implication (2)$\Longrightarrow$(3) was proved in Lemma 3.10, it is sufficient to prove (1)$\Longrightarrow$(2) and (3)$\Longrightarrow$(1).

The proof of Lemma 4.2 below gives a slightly stronger result than implication (3)$\Longrightarrow$(1): with every LRS E, one can explicitly associate a "normalized" **DLIN** program which computes the RAM function Γ represented by E.

LEMMA 4.2. *If a RAM function Γ is linearly represented by an LRS E, then $\Gamma \in$* **DLIN**.

Proof. Let Γ be a RAM function which is linearly represented by the equation E, and let P be the corresponding affine projection. For simplicity, we assume that the type of the input structures of Γ consists of only one function symbol f_{in}. Recall that E consists of one equation of the form $g(x) = \sigma(x)$, where $\sigma(x)$ is a recursion term, and there is a constant c such that the output $s' = \Gamma(s)$ can be extracted from the function $g : [cs.n] \to [cs.n]$ and $s.n$. Instead of formally defining a RAM for Γ, we are going to give an algorithm that can easily be converted into a multimemory RAM, which in turn can be converted into a simple RAM (cf. section 2.2). Its data structure consists of variables p, x and, besides the input structure s and the output structure s', four one-dimensional arrays F_{in}, G, G_{inverse}, and EP. In p, we store the value $cs.n$, which bounds the domain and the range of the function g that is defined by E and s. Next we describe the intended meaning of F_{in}, G, G_{inverse}, and EP. For indices greater than or equal to $s.n$, F_{in} is 0; for smaller indices, it is determined by $s.f$. The main computation proceeds in p rounds, numbered from 0 to $p-1$. After round i of this computation, the following invariants should hold:

(a) for each $j < p$,

$$G[j] = \begin{cases} g(j) & \text{if } j \le i,\\ j & \text{if } i < j < p,\end{cases}$$

(b) $\text{EP}[j] = g^{\leftarrow}(j)$ for each $j \le i$, and

(c) for each $j < p$,

$$G_{\text{inverse}}[j] = \begin{cases} \max\{l \le i \mid g(l) = j\} & \text{if such an } l \text{ exists},\\ p & \text{otherwise.}\end{cases}$$

The computation of the values $G[i]$ is relatively straightforward. We associate with every recursion term $\sigma(x)$ a *programming term* $\sigma[x]$, which is recursively defined as follows:

- if $\sigma(x)$ is 1, n, x, then $\sigma[x]$ is 1, n, x, respectively;
- if $\sigma(x) = g^{\leftarrow}(x - \delta)$ for some δ, then $\sigma[x] = \mathrm{EP}[x - \delta]$;
- if $\sigma(x) = g[\tau(x)]_x$ for some subterm τ, then $\sigma[x] = G[\tau[x]]$;
- if $\sigma(x) = f_{\mathrm{in}}(\tau(x))$ for some subterm τ, then $\sigma[x] = F_{\mathrm{in}}[\tau[x]]$;
- if $\sigma(x) = \tau_1(x) * \tau_2(x)$ for subterms τ_1 and τ_2, then $\sigma[x] = \tau_1[x] * \tau_2[x]$.

In round i, we compute $G[i]$ by evaluating $\sigma[i]$. The only difficulty that arises is the evaluation of subterms of the form $g^{\leftarrow}(x-\delta)$, which would need more than a constant number of steps if it was done in a straightforward manner. Instead, we make use of the array EP, which contains, after round i, the value $g^{\leftarrow}(j)$ for each $j \le i$. G_{inverse} always holds a partial inverse of the function g on $\{0, \ldots, j\}$ and is used to compute EP.

The algorithm for Γ is as follows.

Input s
{ Initializations }
$p := cs.n$
EP[0] := 0
FOR $j := 0$ TO $s.n - 1$ DO
    $F_{\mathrm{in}}[j] := s.f(j)$
FOR $j := s.n$ TO $p - 1$ DO
    $F_{\mathrm{in}}[j] := 0$
FOR $j := 0$ TO $p - 1$ DO
    $G[j] := j$
    $G_{\mathrm{inverse}}[j] := p$
{ Main loop }
FOR $i := 0$ TO $p - 1$ DO
    $G[i] := \sigma[i]$
    $\mathrm{EP}[i] := \min(i, G_{\mathrm{inverse}}[G[i]])$
    $G_{\mathrm{inverse}}[G[i]] := i$
{ Output }
Compute the result s'' by applying the affine projection $P_{s.n}$
to the structure $s' = ([p], G)$

We show by induction on i that the above invariants (a)–(c) hold after each round i. Of course, they are satisfied after the initializations, i.e., before round 0. For the induction step, assume that (a)–(c) are satisfied *before* round i ($0 \le i \le p - 1$). We show that they also hold *after* round i and hence before round $i + 1$.

- $G[i] = g(i)$ holds because, by induction, we have $G[j] = g[j]_i$ for each $j \in [p]$ and $\mathrm{EP}[j] = g^{\leftarrow}(j)$ for each $j < i$; hence bounded application and equal predecessor are evaluated correctly.
- The assignment $\mathrm{EP}[i] := \min(i, G_{\mathrm{inverse}}[G[i]])$ implies that $\mathrm{EP}[i] = g^{\leftarrow}(i)$ by induction.
- It is also straightforward to observe that (c) is maintained by the assignment $G_{\mathrm{inverse}}[G[i]] := i$.

So (a)–(c), in particular (a), are invariants; this implies immediately that the program is correct.

The number of steps to evaluate a recursion term is linear in the size of the term. As the recursion term of E is fixed, we conclude that the computation of each $G[i]$ needs only a constant number of steps; therefore, the overall computation time of the program is $O(s.n)$.

It should be stressed that the structure of the resulting program is very simple, with essentially one main loop. □

It remains to prove the implication (1)$\Longrightarrow$(2) of Theorem 4.1.

LEMMA 4.3. *If a RAM function* Γ *is computed in time* $O(n)$ *on a simple* $\{+,-\}$*-RAM* M*, then* Γ *is linearly represented by some LSRS* S*.*

Proof. Let Γ be a RAM function. For simplicity, we assume that the type of the input as well as the type of the output consist of only one function symbol f. Let M be as in the statement of the lemma, and let $c > 0$ be such that M needs less than cn steps and uses numbers of size less than cn on $c(\Gamma)$-bounded inputs of size n.

The proof is similar to the one that was given to prove Theorem 3.1 (a) in [34]. Let us sketch the main ideas first. We are going to construct an LSRS that uses functions I, A, B, N, R_A, which are intended to describe the situation of the RAM *before* each step x, where

- $I(x)$ holds the current instruction number,
- $A(x)$, $B(x)$, and $N(x)$ hold the current values of registers A, B, N, respectively, and
- $R_A(x)$ holds the value of the register, the address of which is currently contained in register A.

For convenience, we also use functions I', A', B', R'_A, N', which describe the situation *after* a step of the RAM. By case distinctions on the value of $I(x)$, most of the simulation of the computation of M is straightforward. For example, if M executes a statement $A := A + B$, then the equations will force $A'(x) = A(x) + B(x)$. The main complication arises from the instruction $A := R_A$, which loads the content of the register, the number of which is contained in A (before the execution of the instruction), into A. Note that the functions that are defined by S do *not* explicitly encode the values of all registers of M at each time step t but only the content of the register to which A points. To get the right value for $R_A(t)$, the last time step before t at which A contained the value $A(t)$ (i.e., the current value) must be identified. If no such time step exists, we have to refer to the input. Recall that $s.f(i)$ is stored in $R(i)$ at the beginning of the computation. This is the point at which the recursion operation comes into play.

Compared with the proof of [34], we have to do some extra work here, as the recursion schemes in the present article have more restricted operations. In particular, in order to facilitate the simulation of $A := R_A$ instructions, we divide the domains of the functions into three main parts, $\{0, \dots, cn-1\}$, $\{cn, \dots, 2cn-1\}$, and $\{2cn, \dots, 3cn-1\}$. We use the first part to "store" $s.f$, the second part to actually simulate the computation of M, and the third part to extract the output function. In the third part, we use R_A for another purpose than that described before, namely, to encode the output.

We will take the liberty of using definition by cases (cf. Lemma 3.5) and some simple arithmetic operations that are not directly available in an LSRS. The translation into a pure LSRS is straightforward but involves the use of some more functions.

To simplify the presentation, we are going to describe the LSRS separately for the three parts of the domain. These definitions have then to be combined by using definition by cases.

The order of the function symbols is $I, A, B, N, R_A, I', A', B', N', R'_A$.

For $x < cn$, A and R_A are defined by

$$A(x) = x,$$
$$R'_A(x) = f(x).$$

TABLE 1
Equations of S for the middle part $\{cn, \ldots, 2cn-1\}$ of the domain.

$$I(x) = \begin{cases} 1 & \text{if } x = cn, \\ I'(x-1) & \text{otherwise,} \end{cases}$$

$$I'(x) = \begin{cases} i_0 & \text{if } I(x) \text{ is } \textbf{IF } \boldsymbol{A = B} \textbf{ THEN } \boldsymbol{I(i_0)} \textbf{ ELSE } \boldsymbol{I(i_1)} \text{ and } A(x) = B(x), \\ i_1 & \text{if } I(x) \text{ is } \textbf{IF } \boldsymbol{A = B} \textbf{ THEN } \boldsymbol{I(i_0)} \textbf{ ELSE } \boldsymbol{I(i_1)} \text{ and } A(x) \neq B(x), \\ I(x) & \text{if } I(x) \text{ is HALT,} \\ I(x)+1 & \text{otherwise,} \end{cases}$$

$$A(x) = \begin{cases} 0 & \text{if } x = cn, \\ A'(x-1) & \text{otherwise,} \end{cases}$$

$$A'(x) = \begin{cases} c & \text{if } I(x) \text{ is } \boldsymbol{A := c}, \\ A(x) * B(x) & \text{if } I(x) \text{ is } \boldsymbol{A := A * B}, \\ R_A(x) & \text{if } I(x) \text{ is } \boldsymbol{A := R_A}, \\ N(x) & \text{if } I(x) \text{ is } \boldsymbol{A := N}, \\ A(x) & \text{otherwise,} \end{cases}$$

$$B(x) = \begin{cases} 0 & \text{if } x = cn, \\ B'(x-1) & \text{otherwise,} \end{cases}$$

$$B'(x) = \begin{cases} A(x) & \text{if } I(x) \text{ is } \boldsymbol{B := A}, \\ B(x) & \text{otherwise,} \end{cases}$$

$$N(x) = \begin{cases} n & \text{if } x = cn, \\ N'(x-1) & \text{otherwise,} \end{cases}$$

$$N'(x) = \begin{cases} A(x) & \text{if } I(x) \text{ is } \boldsymbol{N := A}, \\ N(x) & \text{otherwise,} \end{cases}$$

$$R_A(x) = R'_A[A^{\leftarrow}(x)]_x,$$

$$R'_A(x) = \begin{cases} B(x) & \text{if } I(x) \text{ is } \boldsymbol{R_A := B}, \\ R_A(x) & \text{otherwise.} \end{cases}$$

All other functions can be defined arbitrarily in the first part of the domain; for concreteness, we define them all to equal 0. The initialization will enable the LSRS to smoothly recover the contents of the register R_A pointed to by A in the second part of the domain by means of the Equal-Predecessor operation.

Table 1 shows the equations of S that define the functions on the middle part $\{cn, \ldots, 2cn-1\}$ of the domain to simulate the computation of M. Note that cn encodes the initial instant of this computation.

We note that the functions I and I' take only a fixed number of values. In the conditions of the above definitions by cases, we use some abbreviations. As an example, "$I(x)$ is $\boldsymbol{A := c}$" is an abbreviation for "$I(x) = i_1$ or $I(x) = i_2$ or ... or $I(x) = i_q$," where the i_j are all numbers of instructions $\boldsymbol{A := c}$. The expression "$I(x)$ is $\textbf{IF } \boldsymbol{A = B} \textbf{ THEN } \boldsymbol{I(i_0)} \textbf{ ELSE } \boldsymbol{I(i_1)}$" has to be interpreted similarly, depending on the possible values of i_0 and i_1. Furthermore, as mentioned before, the constants that occur in the above scheme have to be replaced by additional functions in a pure LSRS. References to $x-1$ are made via recursion. For example, the equation $I(x) = I'(x-1)$ is an abbreviation of $I(x) = I'[\mathbf{1}^{\leftarrow}(x)]_x$. Remember that $\mathbf{1}(i) = 1$ for every i.

As mentioned before, we use R_A to extract the output values of the computation of M, given by the value n' of register N and the content of $R(0), \ldots, R(n'-1)$ at the end of the computation.[7] For $2cn \leq x < 3cn$, the LSRS consists of the equations

$$A(x) = x - 2cn,$$
$$R_A(x) = R'_A[A^{\leftarrow}(x)]_x.$$

All other functions h are defined by $h(x) = h(x-1)$.

It remains to be checked that the equations of S correctly define the intended functions which describe the computation of M and that S also induces the correct RAM output.

For this purpose, let s be a RAM data structure, $\mathbf{1}, \mathbf{n}, \mathbf{id}$, as before, and let I, A, B, N, R_A, I', A', B', N', and R'_A be functions that fulfill the equations of S. First, it is clear that, for all $a < cs.n$, the following hold:

$$A(a) = a,$$
$$R'_A(a) = s.f(a),$$
$$I(a) = B(a) = N(a) = R_A(a) = I'(a) = A'(a) = B'(a) = N'(a) = 0.$$

The values of the functions at $cs.n$ describe the situation of the RAM at the beginning of the computation with respect to the current instruction number and the contents of registers A, B, and N. Furthermore, the definition of A and R'_A on the first part of the domain mirrors the fact that, at the beginning of its computation, M has the value $s.f(i)$ in register $R(i)$.

The proof that the values of the defined functions at $cs.n+t$ encode the situation of M at time t, as intended, is an easy induction on t. The most difficult part is the correctness of function R_A. Note first that, as $A(a) < cs.n$, for every a, the initialization of the first part of the domain ensures that $A^{\leftarrow}(a) < a$ for every $a \in \{cn, \ldots, 2cn-1\}$. In fact, there are 2 cases.

- $cs.n \leq A^{\leftarrow}(a) < a$. In this case, there exists a b, with $cs.n \leq b < a$, such that $A(b) = A(a)$; hence the current R-register has been visited before in the computation of M, and $A^{\leftarrow}(a)$ is the last step for which this happened. Hence $R'_A(A^{\leftarrow}(a))$ gives the correct value of $R_A(a)$.
- $A^{\leftarrow}(a) < cs.n$. In this case, there is no b, with $cs.n \leq b < a$, such that $A(b) = A(a)$; hence the current R-register has not been visited before and should still hold the value $f(A(a))$. By the initialization, it follows that $A^{\leftarrow}(a) = A(a)$ and $R'_A(A(a)) = f(A(a))$, as required.

Finally, we have to show that S correctly defines the output $\Gamma(s)$. By definition of R_A and N on the last part of the universe, it follows analogously that, for each a, $R_A(2cs.n + a)$ contains the content of register $R(a)$ at the end of the computation. Furthermore, $N(3cs.n-1)$ has the value of register N at the end of the computation. Hence, with a suitable projection, $\Gamma(s)$ can be extracted from the functions that are defined by S.

Altogether, we have shown that S correctly defines $\Gamma(s)$. In other terms, Γ is linearly represented by S. □

This completes the proof of Theorem 4.1.

[7] Recall that M uses less than $cn-1$ steps. Therefore, as the first step is simulated at position cn, at position $2cn-2$ the simulation is finished, and we can use the position $2cn-1$ for other purposes, e.g., to encode the length of the output.

5. Complete problems for deterministic linear time. In this section, we are going to exhibit two problems that are related to the recursion schemes defined before and that are complete for **DLIN** under affine reductions. The first problem is an unfolded and nonuniform version of the LSRS algebra. It concerns the simultaneous definition of k functions $f_1, \ldots, f_k : [p] \to [p]$ for some $p > 0$. A *simultaneous function definition (SFD) instance* S consists of a positive integer p, a set $F = \{f_1, \ldots, f_k\}$ of k unary function symbols, and a system of kp equations

$$f_i(a) = (\sigma_i(a))$$

for each $i \in \{1, \ldots, k\}$ and $a \in [p]$, each of one of the following types:

(1) $f_i(a) = b$,

(2) $f_i(a) = h(g(b))$,

(3) $f_i(a) = g(b) * h(c)$,

(4) $f_i(a) = g^{\leftarrow}(b)$,

where $*$ is among $+, -$, b and c are integers in $[p]$, and g, h are in F.

A *valuation* for S is a mapping $V : [p] \times \{1, \ldots, k\} \to [p]$. A valuation is *suitable* if defining all functions f_i by $f_i(a) = V(a, i)$ makes all equations true. Given an SFD instance S and a suitable valuation V, we define the *dependency graph* $G(S, V)$ as follows. The vertices of $G(S, V)$ are all pairs (a, i), where $a \in [p]$ and $i \in \{1, \ldots, k\}$. The edges of (S, V) are defined as follows.

- If $f_i(a) = f_j(f_l(b))$ is an equation of S, then there are edges from (b, l) and $(V(b, l), j)$ to (a, i).
- If $f_i(a) = f_j(b) * f_l(c)$ is an equation of S, then there are edges from (b, j) and (c, l) to (a, i).
- If $f_i(a) = f_j{}^{\leftarrow}(b)$ is an equation of S, then there are edges from (l, j) to (a, i) for all $l \leq b$.

If there is an edge from (b, j) to (a, i), then we also say that $f_j(b)$ is an operand of $f_i(a)$. Note that only edges that correspond to equations of type (2) depend on V. All other edges are determined by S only.

It is not hard to see that an SFD instance S has a valuation V with an acyclic dependency graph if and only if the equations of S can be sorted into a list $E_1, \ldots, E_{kp}$ such that each operand of every equation E_i is defined in an equation E_j with $j < i$. Furthermore, as shown in Lemma 5.1, if this is the case, then the equations have a unique solution.

Example 8. If $k = 2$ and $p = 4$, the following set S of 8 $(= kp)$ sorted equations $E_1, \ldots, E_8$ of the form $f_j(a) = \sigma_j(a)$, $(j \in \{1, 2\}, a \in [p])$, clearly satisfy the required conditions:

- $f_1(2) = 1$;
- $f_2(0) = 3$;
- $f_2(3) = 1$;
- $f_1(0) = f_1(2) + f_2(3)$;
- $f_1(1) = f_2(0) - f_2(3)$;
- $f_1(3) = f_1(f_2(3))$;
- $f_2(1) = f_1{}^{\leftarrow}(3)$;
- $f_2(2) = f_2(1) - f_1(2)$.

Hence $f_1(0) = 1 + 1 = 2$, $f_1(1) = 2$, $f_1(3) = f_1(1) = 2$, $f_2(1) = 1$, (since $f_1(0) = f_1(1) = f_1(3) = 2$ and $f_1(2) = 1 \neq 2$), and $f_2(2) = 0$.

Problem "simultaneous function definition (SFD)".
Input. An SFD instance $S = (p, \{f_1, \dots, f_k\}, (\sigma_i(a))_{i \le k, a < p})$.
Question. Is there a suitable valuation V of S such that $G(S, V)$ is acyclic?
Output. If the answer is positive, the functions $f_1, \dots, f_k$.

In fact, we have defined two problems, one decision problem and one function. The formal representation of the inputs of this problem uses an encoding of a system of equations that is similar to the one that was explained in Example 1 (a). The output is a RAM data structure s' of type $\{C, f\}$ such that $s'.n = kp$, $s'.C = p$, and $s'.f((i-1)p + j) = f_i(j)$ for all $i \in \{1, \dots, k\}$, $j \in [p]$. In the following, unless otherwise stated, we always refer to SFD as a RAM function.

We stick to the convention that the value of an arithmetical expression $g(b) * h(c)$ is 0 in case of overflow, i.e., if its value is not in $[p]$.

Next, we show that the output of an SFD instance is well defined.

LEMMA 5.1. *For each SFD instance S, there is at most one valuation V such that $G(S, V)$ is acyclic.*

Proof. We define inductively a *height* $h(w)$ for vertices w of an acyclic dependency graph as follows. For vertices of in-degree 0, the height is 0. For all other vertices w, $h(w)$ is one plus the maximum height of all vertices u that have an edge to w. The acyclicity ensures that the height is well defined for each vertex. By induction on the height of the vertices, it is easy to show that any two suitable valuations with acyclic dependency graphs are equal. □

THEOREM 5.2. *SFD is complete for* **DLIN** *under affine reductions.*

Proof. We show that, (a) for every RAM function Γ in **DLIN**, we have $\Gamma \le_a$ SFD and (b) SFD is in **DLIN**.

(a) Let $\Gamma \in$ **DLIN** be a RAM function. For simplicity, we shall assume that the type of the input as well as the type of the output consist of only one function f. Let S_1 be the LSRS that linearly defines Γ, as it was constructed in the proof of Lemma 4.3. Given a RAM data structure s as input, we construct an SFD instance S_2 that defines the same functions as S_1. The main idea is to unfold S_1 so that we get one equation for each function of S_1 and each element of the universe $[3cs.n]$. We first describe the construction of a set of equations which allow arbitrary terms $f_i(a)$ and operators from $\{+, -\}$ on the right-hand side. By using more function symbols, this system can be transformed into a pure SFD instance. We describe later how these constructions can be done with affine functions.

The SFD instance S_2 uses all function symbols of the input structure s. Let l be the number of these function symbols. Furthermore, it uses all 10 function symbols of the functions that are defined in S_1. Let c be the constant of the proof of Lemma 4.3. The equations for the symbols of the input structure are simply assignments. For $a \ge s.n$, we assign 0. We construct equations for the three parts $\{0, \dots, cs.n - 1\}$, $\{cs.n, \dots, 2cs.n - 1\}$, and $\{2cs.n, \dots, 3cs.n - 1\}$ separately. For the definition of the second and the third part of R_A, we introduce one new function h, defined by $h(x) = A^{\leftarrow}(x)$. Hence the constructed SFD instance has $k = l + 11$ function symbols, $p = kcs.n$, and each $a \in [cs.n]$ gives rise to $3k$ equations (k equations for each of a, $cs.n + a$, and $2cs.n + a$). For most of the function symbols, the construction is straightforward. The treatment of definition by cases is along the lines of Lemma 3.5. For the second and the third part of R_A, we get two equations, $h(a) = A^{\leftarrow}(a)$ and $R_A(a) = R'_A(h(a))$. It is important to note that the semantics of these two equations together is, in its effect for R_A, the same as that of the original

equation $R_A(a) = R'_A[A^{\leftarrow}(a)]_a$, although the bounded application is replaced by a (nonbounded) composition of functions. This is because the construction of Lemma 4.3 ensures that, for all $a \geq cs.n$, $A^{\leftarrow}(a) < a$.

After transforming S_2 into a pure SFD instance S_3, we get a system of equations of the four allowed forms, such that, for each function symbol f and each of the three parts of the universe, all equations have the same form. The list of these equations is encoded into a RAM data structure with one function symbol g as described in Example 1 (a). Recall that numbers contribute only 1 to the length of the encoding of an equation.

As an example, assume that the sequence of $cs.n$ equations

$$f_3(a) = f_5(a) + f_9(a),$$

where $cs.n \leq a < 2cs.n$, defines f_3 on the middle part of the universe. Each equation has length 18. Let this sequence of equations be the ith such sequence in S_3. The sequence will be encoded by $18cs.n$ consecutive values of $T(s).g$ defined by the following equalities, where $D_i(n) = \sum_{j<i} d_j\alpha_j(n)$ and $b \in [cs.n]$:

- $T(s).g(18b + D_i(s.n)) = 0$, the encoding of the symbol f,
- $T(s).g(1 + 18b + D_i(s.n)) = 12$, the encoding of 3, as the size of the alphabet is 9,
- $T(s).g(2 + 18b + D_i(s.n)) = 2$, the encoding of the opening bracket,
- $T(s).g(3+18b+D_i(s.n)) = b+cs.n+9$, the encoding of the integer $a = b+cs.n$, and so on until finally
- $T(s).g(17 + 18b + D_i(s.n)) = 8$, the encoding of the semicolon.

So this ith part of S_3 clearly satisfies the affine requirements (see (3) in Definition 2.3) with $d_i = 18$ and $\alpha_i(n) = cn$. Moreover, as we saw before at the end of the proof of Lemma 4.3, the output transformation can be given by the affine projection

- $P_{s.n}(s').n = N(3cs.n - 1)$ for the size and
- $P_{s.n}(s').f(a) = R_A(a + 2cs.n)$ for the values.

More precisely, because of the output conventions of SFD, the output $\Gamma(s)$ (of type $\{f\}$) can be extracted from the output $s' = SFD(T(s))$ of type $\{C, f\}$, where $s.C = p = 3cs.n$, by the equalities

$$\begin{aligned}\Gamma(s).n &= P_{s.n}(s').n \\ &= f_i(3cs.n - 1) \qquad \text{if } N \text{ is numbered } f_i \\ &= s'.f(3(i-1)cs.n + (3cs.n - 1)),\end{aligned}$$

and, similarly, if R_A is numbered f_j, for each a,

$$\begin{aligned}\Gamma(s).f(a) &= P_{s.n}(s').f(a) \\ &= f_j(a + 2cs.n) \\ &= s'.f(3(j-1)cs.n + (a + 2cs.n)).\end{aligned}$$

(b) We are going to give a high-level description of an algorithm for SFD and argue that it can be easily transformed into a RAM which solves SFD in linear time.

Let an instance S of the SFD problem be given by p, a set $F = \{f_1, \ldots, f_k\}$, and a set $\{E_1, \ldots, E_{kp}\}$ of equations. The size of S is $\theta(kp)$. We have to show that S can be "evaluated" in $O(kp)$ steps.

We use the following arrays.

- For each $i \leq k$, we use an array E_i such that each $E_i[a]$ is the encoding of the equation which defines $f_i(a)$.

- For each $i \leq k$, we use an array F_i with the intention that, at the end, for each $a < p$, $F_i[a]$ should contain $f_i(a)$.
- For each $i \leq k$, we use a Boolean array Known_i such that $\text{Known}_i[a]$ is true only if $F_i[a]$ contains $f_i(a)$. Initially, all entries in these arrays are set to *false*.
- For each $i \leq k$, we use a variable MinUndef_i such that, at each moment of the computation, MinUndef_i is b if $\text{Known}_i[b]$ is *false*, but $\text{Known}_i[a]$ is *true* for all $a < b$. Initially, all MinUndef_i are 0.
- For each $i \leq k$, we use an array Inv_i such that $\text{Inv}_i[a]$ always contains the maximum $b < \text{MinUndef}_i$, such that $f_i(b) = a$, if such a b exists. Initially, all $\text{Inv}_i[a]$ are represented by p, i.e., a value outside $[p]$.
- For each $i \leq k$ and each $a < p$, we use a list $L_i[a]$ that contains pairs (j, b) such that $f_i(a)$ is an operand in the equation which defines $f_j(b)$, except if this equation is of the form (4).
- For each $i \leq k$ and each $a < p$, we use a list $L_i^{\leq}[a]$ that contains the list of all pairs (j, b) such that the equation $E_j[b]$ is $f_j(b) = f_i{}^{\leftarrow}(a)$. Note that the lists $L_i[a]$ and $L_i^{\leq}[a]$ can be directly computed from S in linear time.
- Furthermore, we use a queue Q, which consists of elements of the form (i, a, b), $a, b \in [p], i \leq k$, where each entry (i, a, b) indicates that $f_i(a) = b$, but this fact has not yet been incorporated to the whole data structure; i.e., in particular, $\text{Known}_i[a]$ is still *false*.

The algorithm is essentially a generalization of topological sorting, in fact a variant of the linear time algorithm for Horn-satisfiability [7].

For the initialization, the algorithm does the following. It first checks that the input is a correct SFD instance. This involves sorting the equations by i and a to check that there is exactly one equation for each i and a. A linear time algorithm for this purpose is described in [19]. Then all entries in Known are set to false, all entries in Inv to p, and all values MinUndef to 0. Furthermore, Q and all lists $L[i, a]$ and $L^{\leq}[i, a]$ are initially empty. For each $j \leq kp$, if equation E_j defines $f_i(a)$, then we set $E_i[a] := E_j$. For each equation of the form $f_i(a) = b$, we add (i, a, b) to Q. For each equation of the form $f_i(a) = f_h(f_j(b))$, we add (i, a) to $L[j, b]$. For each equation of the form $f_i(a) = f_j(b) * f_h(c)$, we add (i, a) to $L[j, b]$ and $L[h, c]$. For each equation of the form $f_i(a) = f_j{}^{\leftarrow}(b)$, we add (i, a) to $L^{\leq}[j, b]$.

The main computation of the algorithm is shown in Table 2.

Finally, it checks that all of the values of the functions $f_1, \ldots, f_k$, as represented by the array F_i, have been computed; i.e., $\text{MinUndef}_i = p$ for each $i \leq k$.

It remains to show that this algorithm is correct and works in linear time. The latter is easy to see. First, for each pair (i, a), at most one triple (i, a, b) is ever added to Q. Hence the outermost WHILE loop is executed at most kp times. Second, the use of MinUndef_i ensures that the innermost WHILE loop is also executed at most kp times. As the time for the initialization and the final tests is also linear, the overall running time is at most linear. The correctness of the algorithm can be shown by an induction on the height of the vertices of the dependency graph that corresponds to S and the valuation that is computed by the algorithm.

Note that the overall size of each of the multidimensional arrays is at most kp. Hence, e.g., the arrays F_i can be simulated[8] by a one-dimensional array F via $F_i[a] = F[ka+i]$. More generally, the algorithm uses only integers of value $O(kp)$, particularly in linear space. □

[8]Note that this RAM algorithm uses multiplication.

TABLE 2
Main part of the evaluation algorithm.

```
WHILE Q is not empty DO BEGIN
    Remove a triple (i, a, b) from Q
    F_i[a] := b; Known_i[a] := true
    For each pair (j, c) in L[i, a] DO BEGIN
        IF E_j[c] is f_j(c) = f_h(f_i(a)) THEN BEGIN
            Set E_j[c] := "f_j[c] = f_h(b)"
            IF Known_h[b] THEN add (j, c, F_h[b]) to Q
                ELSE add (j, c) to L[h, b]
        END
        IF E_j[c] is "f_j(c) = f_i(a) * f_h(d)" THEN
            Set E_j[c] := "f_j[c] = b * f_h(d)"
        IF E_j[c] is "f_j(c) = f_h(d) * f_i(a)" THEN
            Set E_j[c] := "f_j[c] = f_h(d) * b"
        IF E_j[c] is "f_j(c) = f_i(a)" THEN
            add (j, c, b) to Q
        IF E_j[c] is "f_j(c) = f_i(a) * d" THEN
            add (j, c, b * d) to Q
        IF E_j[c] is "f_j(c) = d * f_i(a)" THEN
            add (j, c, d * b) to Q
    END
    WHILE MinUndef_i = a and Known_i[a] DO BEGIN
        For each pair (j, c) in L^≤[i, a] DO
            add (j, c, min(a, Inv_i[F_i[a]])) to Q
        MinUndef_i := a + 1; Inv_i[F_i[a]] := a; a := a + 1
    END
END
```

The SFD function is strongly related to LSRS. Next we describe a problem that is related to LRS and that also turns out to be complete for **DLIN**.

An *inductive function definition (IFD) instance* S consists of a positive integer p and a list of p equations $E_0, \ldots, E_{p-1}$, where each E_a is of one of the following forms:

(i) $f(a) = c$ for some $c \in [p]$,
(ii) $f(a) = f(b) * f(c)$, where $* \in \{+, -\}$ and $b, c < a$,
(iii) $f(a) = f(f^{\leftarrow}(a-1) + 1)$.

Problem "inductive function definition (IFD)".
Input. An IFD instance $S = (p, E_0, \ldots, E_{p-1})$.
Output. The function $f : [p] \to [p]$, as defined by S.

As usual, we stick to the convention that the value of $f(b) * f(c)$ is 0 in case of overflow. As the equations are evaluated in the order in which they are given, and as this order agrees with the natural order on $[p]$, there is no ambiguity in the definition of f. However, it might happen that the output is undefined because $f(b) \neq a - 1$ for each $b < a - 1$, for an equation of type (iii), for some a. In this case, we define the output function to be constant 0.

THEOREM 5.3. *The RAM function IFD is complete for* **DLIN** *under affine reductions.*

Proof. Again we have to show that (a) for every RAM function Γ in **DLIN**, we have $\Gamma \leq_a$ IFD and (b) IFD is in **DLIN**.

Item (b) is much easier to show than in Theorem 5.2, as the equations are simply evaluated in the order in which they appear. This implies that the evaluation of equations of kind (iii) is similarly straightforward but simpler than that in the proof of Theorem 5.2.

It remains to show that IFD is complete for **DLIN**. This proof combines many of the techniques that we have described before in this article. Let $\Gamma \in \mathbf{DLIN}$ be a RAM function that is computed by a safe RAM (cf. section 2). Let S_1 be an LSRS for Γ, as constructed in the proof of Lemma 4.3, in which all definitions by cases, besides those which refer to the partition of the domain into three parts, are arithmetized as sketched in the proof of Lemma 3.5. Furthermore, we allow in S_1 right-hand sides like $I(x-1)$, which can be directly dealt with in the IFD by equations of type (ii). In particular, S_1 uses recursion only once, that is, in $R_A(x) = R'_A[A^{\leftarrow}(x)]_x$ for the middle part and the third part of the domain. Next we transform S_1 into an LRS S_2, similarly as in the proof of Lemma 3.10. The transformation here is much easier because, in S_1, only recursion with Equal-Predecessor over $A(x)$ is used. Let c be as in Lemma 4.3; i.e., all functions will be defined on the domain $[3cs.n]$ for input data structures s. Let c' be a constant such that the evaluation of S_1 always uses only numbers that are smaller than $c's.n$. We make use of an additional function A^+ with the intention that, for all a, $A^+(a) = A(a) + c'n$. Let k be chosen such that the functions of S_1 besides A, R_A, and R'_A can be numbered as $g_3, \ldots, g_{k-1}$. The function f that is defined by S_2 shall encode the functions of S_1 and A^+ in the following way. For each $a \in [3cs.n]$, we get

$$\begin{array}{cc}
f(2ka) = A(a) & f(2ka+k) = A(a) \\
f(2ka+1) = A^+(a) & f(2ka+k+1) = A^+(a) \\
f(2ka+2) = R_A(a) & f(2ka+k+2) = R'_A(a) \\
f(2ka+3) = g_3(a) & f(2ka+k+3) = g_3(a) \\
\vdots\ \vdots\ \vdots & \vdots\ \vdots\ \vdots \\
f(2ka+k-1) = g_{k-1}(a) & f(2ka+k+k-1) = g_{k-1}(a)
\end{array}$$

Now we unfold S_2 with respect to the input data structure s, as described in the following. For simplicity, we assume that s only has one function f'. First, S_3 gets $s.n$ equations $f(a) = s.f'(a)$, where the values $s.f'(a)$ are given by the input and are therefore constants for S_3. Then we have $s.n$ equations of the form $f(a) = c's.n$, where $c's.n$ is also a constant for S_3. The value $c's.n$ is needed in the next step to compute $A^+(x)$ from $A(x)$. Next we unfold S_2 with respect to $3cs.n$ analogously to the proof of Theorem 5.2. Note that all equations are "shifted" by $2s.n$, which can be easily adapted by the coefficients of the affine functions that are constructed. For all equations that do not use recursion, this unfolding is straightforward. Those equations that are obtained from the only recursion equation $R_A(x) = R'_A[A^{\leftarrow}(x)]_x$ of S_2 are encoded by equations of the form $f(2ka+2) = f(f^{\leftarrow}(2ka+1)+1)$. Note that we represented all function values besides those of R_A and R'_A twice in f to make this encoding possible with an equation of type (iii).

The definition of S_3 by affine functions can be done in a similar manner as sketched in the proof of Theorem 5.2. □

6. Conclusion. We have given two algebraic characterizations of **DLIN**. We feel that the existence of such machine-independent characterizations strengthens the intrinsic interest of this complexity class. Moreover, maybe they will help us to find nice logical characterizations of **DLIN** as was done for **NLIN** [20, 29, 8]. The goal of such logical characterizations would be to allow some of the techniques of finite model theory that have been developed to prove nonexpressibility results, to show nonlinear

lower bound results for concrete algorithmic problems. For surveys on this topic see, e.g., [10, 35].

Logical characterizations seem to be more difficult to obtain for deterministic linear time classes than for nondeterministic linear time classes. The known characterizations of deterministic polynomial time suggest two possible approaches. Following Graedel's characterization of **P** by existential second-order Horn logic [13], one could try to get a logical characterization of **DLIN** by restricting the formulas that are used in the characterization of **NLIN**. The other approach would be to use some kind of fixed point operator as in the work of Immerman and Vardi [24, 36]. Although the results of this article can be used to get logical characterizations by both approaches, the resulting logic is too complicated to be of any use.

We have defined a new very strict machine-independent reduction adapted for linear time classes defined by RAMs and at least as restrictive as the classical reductions like reset-log-lin [6], linear time and logarithmic space reductions, etc. Our affine transformation transforms a structure s into a new structure s', which is a kind of copy of s, with possible shuffle and repetitions; in particular, the value of each element $f'(y)$ of the transformed structure s' depends only on one value $f(x)$ of the original structure s.

Moreover, the class of affine transformations is closed under various operations: composition, concatenation, shuffle. Typically, the transformation of $s = ([n], f_0, \ldots, f_{k-1})$ into $s' = ([kn], f')$, where $f' : [kn] \to \mathbb{N}$ is defined by $f'(kx + r) = f_r(x)$ for $x < n$ and $r < k$, is affine.

It is easy to see that an affine reduction is highly parallelizable; more precisely, it is in $\mathbf{NC}^1$, i.e., computable on a uniform Boolean circuit of fan-in two and depth $O(logn)$.

We have shown that affine reductions are the right kind of reductions to define both **DLIN**-completeness and **NLIN**-completeness. We are convinced that they can be applied both in the context of tractable problems, e.g., linear reductions to HORN-SAT, and in the context of intractable problems, e.g., reducibility from SAT or 3-SAT.

Finally, we note that there is another kind of machine-independent reduction, namely, the logical reduction; it would be worthwhile to compare our affine reduction with the logical quantifier-free reduction used by [27] for linear reductions of some **NP**-complete problems, such as KERNEL, to and from problem SAT. We conjecture that an affine reduction between isomorphical invariant problems is also quantifier-free. The converse is unclear.

We have shown that the **NLIN**-complete problem CONTRACT is **NLIN**-complete not only via $\mathbf{DTIME}(n)$-reductions but also via affine reductions. We can also prove the respective statement, left as an exercise to the reader, for the more classical **NLIN**-complete problem reduction of incompletely specified automata (RISA) [11, 16]. We feel that our **DLIN**-complete problems SFD and IFD are still interesting even if they seem to be slightly less natural than CONTRACT and RISA because of the use of the Equal-Predecessor operator. More precisely, IFD appears to be a minimal problem in the following sense: it involves only one function; each one of the three forms (i)–(iii) of the equations of an IFD instance is or seems to be necessary. Of course, on the one hand, (i) $f(a) = c$ and (ii) $f(a) = f(b) * f(c)$ are required to initialize some values and compute new ones, but, on the other hand, (i)–(ii) do not seem to be sufficient unless any **DLIN** algorithm can be implemented in linear time on a write-once RAM, i.e., a RAM which may only write a value once in any register but is not allowed

to erase or rewrite it—an unlikely simulation; see [23] and [32]. Besides, note that IFD does not involve composition of functions; the "execution" of an equation (iii) $f(a) = f(f^{\leftarrow}(a-1)+1)$ has a flavor of "neighborhood copying": search the last $b < a-1$ such that $f(b) = f(a-1)$, and copy the neighbor value $f(b+1)$ into $f(a)$ so that the pairs $(f(b), f(b+1))$ and $(f(a-1), f(a))$ are equal.

If an affine transformation is regarded as a kind of copying process, then any **DLIN** (resp., **NLIN**) problem can be regarded as a subproblem of any **DLIN**-complete (resp., **NLIN**-complete) problem such as IFD (resp., RISA). That means that our above-mentioned linear-time algorithms for IFD and SFD are universal or most general programs with respect to the other linear-time programs, e.g., topological sorting, algorithms for Horn-satisfiability, or for graph planarity, and so forth.

The fact that SFD and IFD are **DLIN**-complete via affine reductions immediately implies that such a problem is at least as hard as any **DLIN** problem for many complexity measures: time and space on Turing machines, Boolean circuit or PRAM complexity, etc., as exemplified by the following equivalences.

PROPOSITION 6.1.

1. *SFD* $\in$ **DTIME**(n) *if and only if* **DLIN** = **DTIME**(n)*;*
2. $SFD \in$ **DSPACE**$(S(n))$ *if and only if* **DLIN** $\subseteq$ **DSPACE**$(S(n))$ *for any space constructible function* $S(n) \geq \log n$*;*
3. *SFD* $\in \mathbf{NC}^1$ *if and only if* **DLIN** $\subseteq \mathbf{NC}^1$*;*
4. *SFD* $\in$ **DTIME** $- PRAM(t(n))$ *if and only if* **DTIME** $- RAM(n)$ *(that is,* **DLIN***) is included in* **DTIME**$-PRAM(t(n))$ *for any honest function* $t(n) < n$*, where* **DTIME** $- PRAM(t(n))$ *denotes the set of problems computable in parallel time* $t(n)$ *on a PRAM which uses only numbers* $O(n)$*.*

It is very difficult to prove complexity lower bounds for natural combinatorial problems on general-purpose models of computation. The **NLIN**-complete problems are among the very few known exceptions: from the inclusions **DTIME**$(n) \subsetneq$ **NTIME**$(n) \subseteq$ **NLIN** [30, 16], one deduces, e.g., that RISA is not in **DTIME**(n). Note that RISA is in **DLIN** if and only if **DLIN** = **NLIN**. We have no idea on how to prove complexity lower bounds, e.g., on Turing machines, for problems such as HORN-SAT or GRAPH-PLANARITY, but we feel that it should be easier to prove that IFD and SFD cannot be reduced to such a problem; e.g., IFD $\leq_a$ HORN-SAT does not hold, and hence HORN-SAT is not complete for **DLIN**. Intuitively, IFD is not a subproblem of HORN-SAT.

Acknowledgments. We would like to thank Clemens Lautemann, Judy Goldsmith, Ken Regan, and Yuri Gurevich for many valuable hints and suggestions. We also thank an anonymous referee for many helpful comments.

REFERENCES

[1] S. BELLANTONI, *Predicative recursion and the polytime hierarchy*, in Feasible Mathematics II, P. Clote and J. Remmel, eds., Birkhäuser Boston, Boston, 1995, pp. 15–29.

[2] S. BELLANTONI AND S. COOK, *A new recursion-theoretic characterization of the poly-time functions*, Comput. Complexity, 2 (1992), pp. 97–110.

[3] S. BLOCH, J. BUSS, AND J. GOLDSMITH, *Sharply Bounded Alternation within P*, Tech. report TR96-011, Electronic Colloquium on Computational Complexity, 1996, available online from http://ftp.eccc.uni-trier.de/pub/eccc/reports/1996/TR96-011/index.html.

[4] A. COBHAM, *The intrinsic computational complexity of functions*, in Proceedings of Logic, Methodology, and the Philosophy of Science, Y. Bar-Hillel, ed., North–Holland, Amsterdam, 1964, pp. 24–30.

[5] K. J. Compton and C. W. Henson, *A uniform method for proving lower bounds on the computational complexity of logical theories*, Annals Pure Appl. Logic, 48 (1990), pp. 1–79.

[6] K. J. Compton and C. Laflamme, *An algebra and a logic for NC^1*, Inform. Comput., 87 (1990), pp. 241–263.

[7] W. F. Dowling and J. H. Gallier, *Linear-time algorithms for testing the satisfiability of propositional Horn formulae*, J. Logic Programming, 1 (1984), pp. 267–284.

[8] A. Durand, *Linear time and the power of one first-order universal quantifier*, Inform. Comput., 174 (2002), pp. 132–142.

[9] H.-D. Ebbinghaus and J. Flum, *Finite Model Theory*, 2nd ed., Springer-Verlag, Berlin, 1995.

[10] R. Fagin, *Easier ways to win logical games*, in Proceedings of the DIMACS Workshop on Finite Models and Descriptive Complexity, AMS, Providence, RI, 1997, pp. 1–32.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability—A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

[12] E. Grädel, *On the notion of linear time computability*, Internat. J. Found. Comput. Sci., 1 (1990), pp. 295–307.

[13] E. Grädel, *Capturing complexity classes by fragments of second order logic*, Theoret. Comput. Sci., 101 (1992), pp. 35–57.

[14] E. Grädel and Y. Gurevich, *Tailoring recursion for complexity*, J. Symbolic Logic, 60 (1995), pp. 952–969.

[15] E. Grandjean, *A natural NP-complete problem with a nontrivial lower bound*, SIAM J. Comput., 17 (1988), pp. 786–809.

[16] E. Grandjean, *A nontrivial lower bound for an NP problem on automata*, SIAM J. Comput., 19 (1990), pp. 438–451.

[17] E. Grandjean, *Invariance properties of RAMs and linear time*, Comput. Complexity, 4 (1994), pp. 62–106.

[18] E. Grandjean, *Linear time algorithms and NP-complete problems*, SIAM J. Comput., 23 (1994), pp. 573–597.

[19] E. Grandjean, *Sorting, linear time and the satisfiability problem*, Ann. Math. Artif. Intell., 16 (1996), pp. 183–236.

[20] E. Grandjean and F. Olive, *Monadic logical definability of np-complete problems*, in Proceedings of the 8th Annual Conference of the EACSL (CSL 94), Kazimierz, Poland, 1994, Lecture Notes in Comput. Sci. 933, Springer-Verlag, New York, 1995, pp. 190–204.

[21] E. Grandjean and F. Olive, *Monadic logical definability of nondeterministic linear time*, Comput. Complexity, 7 (1998), pp. 54–97.

[22] Y. Gurevich, *Algebras of feasible functions*, in Proceedings of the 24th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1983, pp. 210–214.

[23] Y. Gurevich and S. Shelah, *Nearly linear time*, in Logic at Botik '89, Lecture Notes in Comput. Sci. 363, A. Meyer and M. Taitslin, eds., Springer-Verlag, New York, 1989, pp. 108–118.

[24] N. Immerman, *Relational queries computable in polynomial time*, Inform. Control, 68 (1986), pp. 86–104.

[25] R. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–103.

[26] C. Lautemann, N. Schweikardt, and T. Schwentick, *A logical characterization of nondeterministic linear time on Turing machines*, in Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1563, Springer-Verlag, New York, 1999, pp. 143–152.

[27] C. Lautemann and B. Weinzinger, *Monadic NLIN, or weak logical reductions to SAT*, in Proceedings of the 8th Annual Conference of the EACSL (CSL 99), Madrid, Spain, Lecture Notes in Comput. Sci. 1683, Springer-Verlag, New York, 1999.

[28] D. Leivant, *Ramified recurrence and computational complexity* I: *Word recurrence and poly-time*, in Feasible Mathematics II, P. Clote and J. Remmel, eds., Birkhäuser Boston, Boston, 1995, pp. 320–343.

[29] F. Olive, *A conjunctive logical characterization of nondeterministic linear time*, in Proceedings of the 11th Annual Conference of the EACSL (CSL 97), Aarhus, Denmark, 1997, Lecture Notes in Comput. Sci. 1416, Springer-Verlag, New York, pp. 360–372.

[30] W. J. Paul, N. Pippenger, E. Szemerédi, and W. T. Trotter, *On determinism versus non-determinism and related problems (preliminary version)*, in Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science, Tucson, AZ, 1983, pp. 429–438.

[31] K. Regan, *Machine models and linear time complexity*, SIGACT News, 24 (1993), pp. 5–15.
[32] K. Reinhardt, *Strict sequential P-completeness*, in Proceedings of the 14th Symposium on Theoretical Aspects of Computer Science, 1997, Springer-Verlag, New York, 1997, pp. 329–338.
[33] C. P. Schnorr, *Satisfiability is quasilinear complete in NQL*, J. ACM, 25 (1978), pp. 136–145.
[34] T. Schwentick, *Algebraic and logical characterizations of deterministic linear time classes*, in Proceedings of the 14th Symposium on Theoretical Aspects of Computer Science, Springer-Verlag, New York, 1997, pp. 463–474.
[35] T. Schwentick, *Descriptive complexity, lower bounds and linear time*, in Proceedings of the 12th Annual Conference of the EACSL (CSL 98), Brno, Czech Republic, 1998, Lecture Notes in Comput. Sci. 1584, Springer-Verlag, New York, 1999, pp. 9–28.
[36] M. Y. Vardi, *The complexity of relational query languages*, in Proceedings of the 14th ACM Symposium on Theory of Computing, ACM, New York, 1982, pp. 137–146.

SIAM J. COMPUT.
Vol. 32, No. 1, pp. 231–252

IMPROVED APPROXIMATIONS OF CROSSINGS IN GRAPH DRAWINGS AND VLSI LAYOUT AREAS*

GUY EVEN†, SUDIPTO GUHA‡, AND BARUCH SCHIEBER§

Abstract. We give improved approximations for two classical embedding problems: (i) minimizing the number of crossings in a drawing on the plane of a bounded degree graph; and (ii) minimizing the VLSI layout area of a graph of maximum degree four. These improved algorithms can be applied to improve a variety of VLSI layout problems. Our results are as follows. (i) We compute a drawing on the plane of a bounded degree graph in which the sum of the numbers of vertices and crossings is $O(\log^3 n)$ times the optimal minimum sum. This is a logarithmic factor improvement relative to the best known result. (ii) We compute a VLSI layout of a graph of maximum degree four in a square grid whose area is $O(\log^4 n)$ times the minimum layout area. This is an $O(\log^2 n)$ improvement over the best known long-standing result.

Key words. approximation algorithm, crossing number, graph drawing, VLSI layout

AMS subject classifications. 68W25, 05C85, 68W35

PII. S0097539700373520

1. Introduction.

In this paper, we study two related problems: (1) drawing a bounded degree graph on the plane with the fewest number of crossings of edges and (2) minimization of the VLSI layout area of a graph of maximum degree four in a grid with constant aspect ratio. Considerable attention has been devoted to these problems in the past (see, e.g., [L80, V81, BL84, U84, SSSV97, LR99]).

A *drawing of a graph on the plane* is an injection of the vertices of the graph to points in the plane and a mapping of the edges to simple continuous curves between the vertices' images. A curve may not contain an image of a vertex as an internal point, and three (or more) curves may intersect only at an image of a vertex. A *crossing* is an intersection of two curves at a point that is not an image of a vertex. The *size* of a drawing is the sum of the numbers of vertices and crossings in the drawing. The problem of determining the minimum size of a drawing of a graph on the plane is NP-complete [GJ83].

Bhatt and Leighton, in [BL84], apply a $B(n)$-approximate bisection procedure recursively to decompose a bounded degree graph with n vertices. They prove that this recursive decomposition induces a drawing of size $O(B^2(n)\log^2 n)$ times the minimum size drawing. Shahrokhi et al. [SSSV97] considered straight line drawings induced by decomposition trees and presented a simpler construction of a drawing of the same size as the one achieved in [BL84]. Leighton and Rao [LR99] showed that the above result can be realized with a $(\frac{1}{3}, \frac{2}{3})$-separator. Leighton and Rao also showed how

*Received by the editors June 12, 2000; accepted for publication (in revised form) August 14, 2002; published electronically January 3, 2003. Part of this work was done while the first two authors visited IBM T.J. Watson Research Center. An extended abstract of this paper appeared in *Proceedings of the* 32*nd Annual ACM Symposium on Theory of Computing*, Portland, OR, 2000.
http://www.siam.org/journals/sicomp/32-1/37352.html

†Department of Electrical Engineering, Tel-Aviv University, Tel-Aviv 69978, Israel (guy@eng.tau.ac.il).

‡Computer and Information Science Department, University of Pennsylvania, Philadelphia, PA 19104 (sudipto@cis.upenn.edu). This work was done when the author was a graduate student at Stanford University with research supported by an IBM cooperative fellowship, ARO MURI grant DAAH04-96-1-0007, and NSF award CCR-9357849.

§IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (sbar@watson.ibm.com).

to find such a separator with size bounded by $\alpha(n) = O(\log n)$ times the optimal bisector. This implied an $O(\log^4 n)$- (or $O(\alpha^2(n) \log^2 n)$-) approximation algorithm for the drawing size of a bounded degree graph. The bound on the approximation factor in this algorithm relies only on the fact that an optimal drawing induces a planar graph which admits small vertex separators. The reason that the approximation algorithm applies only to bounded degree graphs is that, in bounded degree graphs, the vertex separators and the edge separators have roughly the same size. It appears that the ideas of [PSS96, SSSV97] may extend to arbitrary degree graphs. Motivated by graph layout problems, we focus on bounded degree graphs for results related to crossing numbers of graphs and on graphs of maximum degree four for results related to layouts.

A *VLSI layout* of a graph of maximum degree four is an embedding of the graph in a grid. The vertices of the graph are injected into the grid vertices, and the edges of the graph are mapped into edge disjoint paths in the grid. In fact, we consider the more restrictive "Manhattan" model of routing (in which layer assignment is trivial) and not the "knock-knee" model. Namely, an intersection of two paths (i.e., images of edges) at an internal grid vertex is allowed provided that one path traverses the grid vertex horizontally and the other path traverses the grid vertex vertically. The objective in the VLSI layout area problem is to minimize the area of the grid. Note that to be able to embed a graph in a grid, it must be of maximum degree four. From now on, we consider only graphs of maximum degree four for the VLSI layout problem.

Leiserson [L80] and Valiant [V81] showed that every planar graph can be embedded in a grid with constant aspect ratio of size $O(n \log^2 n)$. Bhatt and Leighton suggested embedding a graph of maximum degree four in two steps. First, draw it on the plane, and then apply the approximation algorithm for VLSI layouts of planar graphs to the planar graph resulting from the drawing by adding vertices in crossings. The optimal layout gives a feasible drawing of the same size; therefore, the $O(\log^4 n)$-approximation algorithm for drawing graphs on the plane combined with the $O(\log^2 n)$-approximation algorithm for VLSI layouts of planar graphs yield an $O(\log^6 n)$-approximation algorithm for the VLSI layout problem.

A *decomposition tree* of a graph $G = (V, E)$ is a tree with a mapping of the tree nodes to subsets of vertices as follows. The root is mapped to V; every two siblings are mapped to subsets that constitute a partitioning of the subset of V to which their parent is mapped; and leaves are mapped to subsets containing a single vertex. The cut associated with an internal tree node t is the set of edges between the subsets to which the children of t are mapped. Bhatt and Leighton, in [BL84], proposed a special type of decomposition trees called *bifurcators*. These decomposition trees are binary trees, and the cut sizes associated with their nodes decrease exponentially as a function of the depth of the node. Bhatt and Leighton [BL84] demonstrated that a $\sqrt{2}$-bifurcator can be used for a wide variety of problems in VLSI layouts: minimizing capacitive delay, producing fault tolerant layouts, layouts for graphs using prefabricated chips, regular layouts, and layouts minimizing wire crossing to name a few. (The detailed description of these applications is provided in [BL84].) Using the approximation of drawing size, in retrospect, Bhatt and Leighton provided an $O(\log^{2.5} n)$-approximation for the optimal $\sqrt{2}$-bifurcator.

Our results. In this paper, we provide improved approximation algorithms for the above problems. We first provide an $O(\log^3 n)$-approximation algorithm for the drawing size problem. As a consequence of this result, based on Bhatt and Leighton's results, we obtain an $O(\log^2 n)$-approximation for the optimal $\sqrt{2}$-bifurcator problem

and a corresponding improvement for all of its applications.

Our $O(\log^3 n)$-approximation for the drawing size problem combined with layout algorithms for planar graphs yield an $O(\log^5 n)$-approximation for the minimum VLSI layout area. Using further structural properties of the decomposition tree computed in our algorithm, we show that the VLSI layout area problem can be approximated to a factor of $O(\log^4 n)$.

In terms of the best approximation factor known for separators, our results can be stated as follows. Let $\alpha(n)$ denote the smallest known ratio of the separator size (that can be computed efficiently) to the optimal bisector size. We show an $O(\alpha^2(n) \log n)$-approximation of the minimum drawing size and an $O(\alpha^2(n) \log^2 n)$-approximation of the minimum VLSI layout area.

Our approximation algorithms construct a decomposition tree that can be viewed as an approximation of a decomposition tree $\widetilde{T}$ obtained by recursively bisecting the planar graph induced by an optimal drawing. Such a decomposition tree $\widetilde{T}$ induces a drawing of size $O(\log n)$ times the optimal drawing size. In addition, $\widetilde{T}$ can be used in Leiserson's embedding algorithm [L80] to compute a layout of area $O(\log^2 n)$ times the optimal layout area.

The decomposition tree that we compute mimics the useful properties of $\widetilde{T}$. In the problem of drawing a graph on the plane, we attach an estimator $\phi(t)$ to every tree node that estimates the optimal drawing size of the corresponding subgraph. The estimator quality is one-sided; it may not surpass (twice) the drawing size but might be much smaller than the drawing size. The estimators have the following two properties that approximate the properties of the drawing sizes of the subgraphs in $\widetilde{T}$: (a) the cut sizes are bounded by the square-root of the corresponding estimators up to a logarithmic error term; and (b) the estimators decrease exponentially as one goes down the tree. The main difficulty in constructing such an "approximated" decomposition tree with estimators is that the only tool we have is approximate separators. To overcome this difficulty, we apply a top-down approach with rebalancing to guarantee that the estimators decrease exponentially.

Our technique extends to a framework for computing decomposition trees with estimators of other superadditive functions. (Superadditive functions are functions in which the sum of the function values on disjoint subgraphs is no more than the value of the function on the whole graph.) Guha [G00] applied this technique to the fill-in function, yielding better approximations for chordal completion, operation count, and certain cases of elimination height and pathwidth.

For the grid embedding algorithm, we follow the paradigm suggested by Leiserson [L80]. First, we present a "weighted" version of Leiserson's algorithm that can be implemented when the separation properties of the decomposition tree are given relative to *weights* of subgraphs rather than *sizes* of subgraphs. A tempting approach is to use the decomposition tree computed for approximating the drawing size together with the estimators associated with its nodes as weights for the implementation of Leiserson's algorithm. However, this approach fails since the estimators are not adequate as weights. This is because they do not satisfy a superadditivity property. Therefore, we compute new node weights. These weights require a rebalancing of the decomposition tree before the weighted version of Leiserson's algorithm can be applied.

Organization. In section 2, we define the problems and describe the basic tools: decomposition trees, separators, etc. In section 3, we describe decomposition trees with estimators and show that they are useful for drawing graphs. In section 4, we present an algorithm for computing decomposition trees with estimators. In section 5,

an improved approximation of $\sqrt{2}$-bifurcators is presented. In section 6, we present an algorithm for approximating the minimum VLSI layout area.

2. Preliminaries.

Minimum drawing size of a graph. A *drawing* D of a graph G is a one-to-one mapping of the vertices to points on the plane. We call the image of a vertex its *position*. Every edge is mapped to a simple (non-self-intersecting) continuous curve connecting the positions of the end-points of the edge. A curve corresponding to an edge may not contain a position of a vertex as an interior point. A *crossing* is an intersection of the interiors of two curves corresponding to images of edges. (Note that a position of a vertex does not count as a crossing since a position of a vertex cannot be in the interior of a curve.) We do not allow more than two curves to intersect at every crossing. A *point* in a drawing is either a position of a vertex of the graph or a crossing. We denote the minimum number of points, over all drawings of G, by $\mathrm{MDS}(G)$. Note that $\mathrm{MDS}(G) = n + CR(G)$, where $CR(G)$ denotes the (minimum) crossing number of G, and n is the number of vertices in G.

The minimum drawing size of a graph satisfies the following *superadditivity* property.

PROPOSITION 1. *If the vertex set of G is partitioned into disjoint sets and the respective graphs induced by them are denoted by $\{G_i\}_i$, then* $\mathrm{MDS}(G) \geq \sum_i \mathrm{MDS}(G_i)$.

Minimum layout area of a graph. A *VLSI layout* of a graph of maximum degree four is an embedding of the graph in a grid. The embedding consists of an injection of the graph vertices to the grid vertices and a mapping of the graph edges into edge disjoint paths in the grid. The paths into which two edges are mapped may intersect at an internal grid vertex, provided that one edge is mapped to the two horizontal grid edges touching this vertex and the other edge is mapped to the vertical grid edges touching it. The *aspect ratio* of a (rectangular) grid is the ratio of its short dimension to its long one. Let $\textsc{Area}_\sigma(G)$ denote the minimum area required to embed a graph G in a host grid with aspect ratio σ. Let $\textsc{Area}(G) = \min_{\sigma>0} \textsc{Area}_\sigma(G)$. Using a "folding" argument, Leiserson [L80] showed that $\textsc{Area}_1(G) \leq 3 \cdot \textsc{Area}(G)$. It follows that considering only embeddings in squares has only a small effect on the minimum area.

Since every layout is also a drawing, and the area of the layout is an upper bound on the size of the drawing, the following property holds.

PROPOSITION 2. *For every graph of maximum degree four,* $\textsc{Area}(G) \geq \mathrm{MDS}(G)$.

Decomposition trees. A (binary) decomposition tree[1] T of $G = (V, E)$ is a rooted binary tree T, and a mapping of the tree nodes to subsets of vertices is as follows: The root is mapped to V; every two siblings are mapped to subsets that constitute a partition of the subset of V to which their parent is mapped; and leaves are mapped to subsets containing a single vertex. Let V_t denote the set of vertices to which the tree node t is mapped, and let $n_t = |V_t|$. Let $G_t = (V_t, E_t)$ denote the subgraph of G induced by V_t. To each internal tree node $t \in T$, we associate the cut between the vertex sets to which the two children of t are mapped. Formally, let $cut(V_1, V_2)$ denote the edges between vertices in V_1 and vertices in V_2. The cut corresponding to an internal node t is denoted by cut_t. This cut is defined by $cut_t = cut(V_{t_\ell}, V_{t_r})$, where t_ℓ and t_r are the children of t. For an edge $e = (u, v) \in E$, let $t(e)$ be the tree node for which $e \in cut_{t(e)}$. Note that $t(e)$ is the lowest tree node that is mapped to

[1]Decomposition trees may not be binary. However, in this paper, we consider only binary decomposition trees.

a set containing both u and v.

Drawings induced by decomposition trees. In [BL84], Bhatt and Leighton considered (recursive) drawings that are induced by decomposition trees. Shahrokhi et al. [SSSV97] described simple drawings that are induced by ordered decomposition trees.[2] Following Shahrokhi and Shi [SS00], we call this drawing a *linear drawing*. An ordered decomposition tree induces a permutation of the vertices by considering the order of the leaves in a preorder traversal. (Recall that each such leaf is mapped to a subset containing just one vertex in G.) Let $v_1, \ldots, v_n$ denote the vertices of G in this order. Map the vertices to points along a line (also called the *spine*), unit distances apart, according to their order. All of the edges are drawn on one side of the spine as half circles; namely, each edge $e = (v_i, v_j)$ is drawn as half a circle with diameter $|i - j|$ and end-points at v_i and v_j.

Shahrokhi et al. [SSSV97] count the number of crossings in such an induced drawing using the following observation.

LEMMA 3. *In a linear drawing, if the curves corresponding to edges $e = (v_i, v_j)$ and $e' = (v_k, v_\ell)$ cross, then either $t(e)$ is an ancestor of $t(e')$ or $t(e')$ is an ancestor of $t(e)$. (A node is considered both an ancestor and a descendant of itself.)*

Proof. Observe that, for each tree node t, the vertices in V_t are drawn contiguously along the spine. From the drawing it follows that whenever two edges $e = (v_i, v_j)$ and $e' = (v_k, v_\ell)$ cross, then either $k < i \leq \ell \leq j$ or $i \leq k \leq j < \ell$. The lemma follows. □

A crossing of the curves of e and e' is charged to edge e if $t(e)$ is an ancestor of $t(e')$ and to edge e' otherwise. From Lemma 3 it follows that a crossing of the curves of $e = (u, v)$ and e' is charged to e if $t(e')$ is either on the path connecting $t(e)$ to u in T or on the path connecting $t(e)$ to v. Recall that, for an internal tree node t, the number of edges e with $t(e) = t$ is $|cut_t|$. We get the following bound on the number of crossings charged to an edge e.

COROLLARY 4. *Let $P(u, v)$ denote the set of nodes in T on the path from the leaf mapped to u to the leaf mapped to v. The number of crossings in a linear drawing that are charged to the edge $e = (u, v)$ is bounded by $\sum_{t \in P(u,v)} |cut_t|$.*

Shahrokhi et al. [SSSV97] also suggested placing the vertices on a circle (or on the corners of a convex polygon) using the same order and drawing the edges with straight lines. The number of crossings in this drawing also satisfies Corollary 4.

Existence of special separators. In our algorithm, we need to compute a *simultaneous* edge separator. Such an edge separator is a cut that partitions a graph in a balanced way according to *two* measures—the number of vertices and their weights. Suppose that we are given a graph G with vertex weights $w(v)$. For a set of vertices S, let $w(S)$ denote the sum of the weights of vertices in the set S.

DEFINITION 1. *A cut $(S, V - S)$ is a $(\frac{1}{4}, \frac{3}{4})$*-separator with respect to the number of vertices *if* $\min\{|S|, |V - S|\} \geq \frac{1}{4}|V|$.

A cut $(S, V - S)$ is a $(\frac{1}{3}, \frac{2}{3})$-separator with respect to the vertex weights *if* $\min\{w(S), w(V - S)\} \geq \frac{1}{3}w(V)$.

A cut $(S, V - S)$ is a simultaneous (edge) separator *if it is a $(\frac{1}{4}, \frac{3}{4})$-separator with respect to the number of vertices and a $(\frac{1}{3}, \frac{2}{3})$-separator with respect to the vertex weights.*

The following lemma shows that a small simultaneous separator exists provided that vertex weights are not too big.

[2] An ordered tree is a tree in which the children of each internal node are ordered.

LEMMA 5. *If the maximum vertex weight is bounded by $\frac{2}{3}$ of the total weight, then there exists a cut of size $O(\sqrt{\mathrm{MDS}(G)})$ that is a simultaneous edge separator.*

Proof. First, for the sake of completeness, we prove the weighted version of the Planar Separator Theorem [LT79]. Namely, we show that there exists a cut of size $O(\sqrt{\mathrm{MDS}(G)})$ that separates G into subgraphs of weight between one third and two thirds of the total weight. Let $G = (V, E)$. Consider an optimal drawing of G, and let $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ denote the planar graph that is obtained by introducing vertices in the crossings of the optimal drawing. Assign zero weights to vertices in $\widetilde{V} - V$, and keep the weights of vertices in V unchanged. Note that $|\widetilde{V}| = \mathrm{MDS}(G)$. Construct an ordered decomposition tree of $\widetilde{G}$ by applying the Planar Separator Theorem recursively and setting the heavier subgraph as the left child in each step. Let $v_1, \ldots, v_n$ denote the order induced by a preorder traversal on the vertices in V. Define $S_i = \{v_1, \ldots, v_i\}$ and $S_0 = \phi$. Define ℓ as follows:

$$\ell = \min\{i : w(S_i) \geq w(V)/3\}.$$

Observe that $w(V)/3 \leq w(S_\ell) \leq 2w(V)/3$. If $\ell = 1$, then this follows from the fact that $w(v) \leq 2w(V)/3$ for every $v \in V$. If $\ell > 1$, then $w(v_\ell) \leq w(S_{\ell-1}) \leq w(V)/3$ because of the ordering rule, in which the heavier subgraph is set as the left child.

We now prove that the size of the cut $(S_\ell, V - S_\ell)$ is $O(\sqrt{\mathrm{MDS}(G)})$. Let $t_0, t_1, \ldots, t_k$ denote the path in the decomposition tree from the root to the leaf corresponding to v_ℓ. The size of the cut $(S_\ell, V - S_\ell)$ is bounded by $\sum_{i=0}^{k} |cut_{t_i}|$. The Planar Separator Theorem implies that (a) $|cut_{t_i}| \leq O(\sqrt{|\widetilde{V}_{t_i}|})$; and (b) $|\widetilde{V}_{t_i}| \leq (2/3)^i \cdot |\widetilde{V}|$ for every $i = 0, 1, \ldots, k$. This completes the proof of the weighted version of the Planar Separator Theorem. We note that since all of the vertices in $\widetilde{V}$ have constant degree, the weighted version of the Planar Separator Theorem follows from a general theorem of Gazit and Miller [GM90]. We detailed the proof for the sake of completeness and also since a similar construction is used later.

A simultaneous separator can be shown to exist as follows. Let $(S, V - S)$ denote a $(\frac{1}{3}, \frac{2}{3})$-separator with respect to the weights $w(v)$. If $|S|, |V| - |S| \geq \frac{1}{4}|V|$, we are done. Otherwise, without loss of generality, $|S| > \frac{3}{4}|V|$. Let $(S_1, S - S_1)$ denote a $(\frac{1}{3}, \frac{2}{3})$-separator of S with respect to the number of vertices. Without loss of generality, assume that $w(S_1) \leq w(S - S_1)$. In case $w(S - S_1) \geq \frac{1}{3}w(V)$, we are done since the partition into $S - S_1$ and $V - (S - S_1)$ is a good partition and the size of the cut is $O(\sqrt{\mathrm{MDS}(G)})$.

If $w(S - S_1) < \frac{1}{3}w(V)$, let $(S_2, V - S - S_2)$ denote a $(\frac{1}{3}, \frac{2}{3})$-separator of $V - S$ with respect to the weights. Without loss of generality, assume that $w(S_2) \leq w(V - S - S_2)$. We claim that the partition into $(S - S_1) \cup S_2$ and $(V - S - S_2) \cup S_1$ is a good partition. First, note that the size of the cut is $O(\sqrt{\mathrm{MDS}(G)})$. Clearly, it is a $(\frac{1}{4}, \frac{3}{4})$ partition with respect to the number of vertices since $|S_1|, |S - S_1| \geq \frac{1}{4}|V|$.

We show that $\frac{1}{3}w(V) \leq w(V - S - S_2) + w(S_1) \leq \frac{2}{3}w(V)$, and thus it is a $(\frac{1}{3}, \frac{2}{3})$ partition with respect to the weights. The lower bound is proved as follows:

$$\begin{aligned} w(V - S - S_2) + w(S_1) &\geq \frac{1}{2}w(V - S) + w(S_1) \\ &= \frac{1}{2} \cdot [(w(V) - w(S - S_1)) + w(S_1)]. \end{aligned}$$

Since $w(V) - w(S - S_1)$ is at least $\frac{2}{3}w(V)$, the lower bound follows. Also,

$$w(V - S - S_2) + w(S_1) \leq \frac{2}{3}w(V - S) + \frac{1}{2}w(S) \leq \frac{2}{3}w(V) - \frac{1}{6}w(S) \leq \frac{2}{3}w(V). \qquad \square$$

Computing simultaneous separators. The proof of Lemma 5 can be made into an efficient algorithm if one could efficiently compute balanced cuts with respect to vertex weights (or number of vertices). The size of the cuts needs to be $O(\sqrt{\mathrm{MDS}(G)})$. In this section, we present a weaker result; namely, the cut size is $O(\sqrt{\mathrm{MDS}(G)} \cdot \log n)$. This additional $O(\log n)$ factor is also added to the size of the simultaneous separator that we can compute efficiently. The algorithm is based on applying at most twice the Leighton–Rao bicriteria approximation algorithm for separators.

The Leighton–Rao separator algorithm [LR99] finds separators in unweighted graphs as well as weighted graphs. A cut $cut(U, V-U)$ is a b-balanced cut if the weights of U and $V-U$ are at most $1-b$ times the total weight. The Leighton–Rao algorithm receives two balance parameters $b \leq \frac{1}{2}$ and $b' \leq \min\{b, \frac{1}{3}\}$ and returns a b'-balanced cut the size of which is $O(\frac{\log n}{b-b'} \cdot \mathrm{SEP}_b)$, where SEP_b denotes the size of an optimal b-balanced cut.

We describe how to compute a b-balanced cut with respect to vertex weights provided that $b \leq \frac{1}{3}$. (Computing balanced cuts with respect to the number of vertices is simply the case of uniform weights.) The size of the cut is $O(\sqrt{\mathrm{MDS}(G)} \cdot \log n)$.

Set $b' = 1 - \sqrt{1-b}$, and apply the Leighton–Rao algorithm. The partitioning yields a b'-balanced cut $cut(U, V-U)$ with respect to vertex weights. If this cut happens to be also b-balanced, then we are done. Otherwise, assume U is the heavier part (i.e., $w(U) \geq w(V-U)$). Since $cut(U, V-U)$ is not b-balanced, it follows that $w(U) > (1-b)w(V)$. Apply the Leighton–Rao algorithm to U to obtain a b'-balanced cut $(U_1, U-U_1)$. Assume U_1 is the heavier part. Then the cut $cut(U_1, V-U_1)$ is b-balanced. The upper bound follows since $w(U_1) \leq (1-b')w(U) \leq (1-b')^2 w(V) = (1-b)w(V)$. The lower bound follows since $w(U_1) \geq \frac{1}{2}w(U) \geq \frac{1}{2}(1-b) \cdot w(V) \geq b \cdot w(V)$. (The last inequality holds since $b \leq \frac{1}{3}$.) Note that the sizes of both cuts are bounded by $O(\sqrt{\mathrm{MDS}(G)} \cdot \log n)$ since the sizes of the optimal b-balanced cuts of V and U are bounded by $\sqrt{\mathrm{MDS}(G)}$.

Another way to compute simultaneous separators with "relaxed" balance parameters uses the spreading metric based algorithm [ENRS99] for simultaneous separators.

Weight functions induced by cuts. The weight functions used in our decomposition algorithm are defined by cuts. We consider two subsets of vertices $A, B \subseteq V$ and call A the *home* set and B the *outside* set. The weight of $v \in A$ with respect to its *home* set A and the *outside* set B, denoted $W_{A,B}(v)$, is defined to be the number of edges connecting v with vertices in $B - A$. In other words, $W_{A,B}(v)$ equals the number of edges in $cut(A, B-A)$ incident to v. Note that $\sum_{v \in A} W_{A,B}(v) = |cut(A, B-A)|$. Since the graphs are of bounded degree, the weight function is bounded as well. We note that for some weight functions $W_{A,B}$, there may be a vertex v such that $W_{A,B}(v) > \frac{2}{3}\sum_{v \in A} W_{A,B}(v)$. Consequently, we cannot apply Lemma 5 to find a simultaneous separator in such cases. To keep the presentation simpler, we first assume that such cases of unbalanced weight functions do not happen and later note how to cope with them.

3. A decomposition tree for minimizing the drawing size.

3.1. Motivation. Following Bhatt and Leighton, the drawings that we obtain are induced by decomposition trees. Specifically, these drawings are linear drawings as defined by Shahrokhi et al. [SSSV97]. The definition of decomposition trees used for approximating the minimum drawing size is motivated by an "ideal" decomposition tree. Loosely speaking, this ideal decomposition tree is obtained by applying the Planar Separator Theorem recursively as in the proof of the weighted version

of the Planar Separator Theorem in Lemma 5. More formally, the ideal decomposition tree T_I is defined as follows. Let $\widetilde{G}$ denote a planar graph that is obtained by introducing vertices in the crossings of an optimal drawing of $G = (V, E)$. The decomposition tree $\widetilde{T}$ of $\widetilde{G}$ is obtained by recursively separating $\widetilde{G}$ using the Planar Separator Theorem. The decomposition tree $\widetilde{T}$ induces the decomposition tree T_I of G as follows: (a) For every $t \in \widetilde{T}$, define $V_t = \widetilde{V}_t \cap V$. (b) Prune the children, if any, of every tree node t for which $|V_t| = 1$.

The size of a linear drawing induced by an ideal decomposition tree T_I is analyzed as follows. For every internal tree node $t \in T_I$, $|cut(V_t, V - V_t)| \leq |cut(\widetilde{V}_t, \widetilde{V} - \widetilde{V}_t)|$. The Planar Separator Theorem implies that $|cut(\widetilde{V}_t, \widetilde{V} - \widetilde{V}_t)| = O(\sqrt{|\widetilde{V}_t|})$. By Corollary 4, in the linear drawing induced by T_I, an edge $e = (u, v)$ is charged for at most $\sum_{t \in P(u,v)} |cut(V_t, V - V_t)| = \sum_{t \in P(u,v)} O(\sqrt{|\widetilde{V}_t|})$ crossings. The recursive separation implies that $|\widetilde{V}_t|$ decreases exponentially as one goes down the tree; therefore, an edge $e = (u, v)$ is charged for $O(\sqrt{|\widetilde{V}_{t(e)}|})$ crossings. To bound the total charges, charge a tree node t for all of the edges e for which $t = t(e)$. Hence the charging of a tree node t is bounded by $O(|cut(\widetilde{V}_t, \widetilde{V} - \widetilde{V}_t)| \cdot \sqrt{|\widetilde{V}_t|}) = O(|\widetilde{V}_t|)$. The tree nodes in the same layer of $\widetilde{T}$ induce a partition of $\widetilde{G}$. It follows that the sum of $|\widetilde{V}_t|$ over all tree nodes t in the same layer is bounded by $O(|\widetilde{V}|) = O(\mathrm{MDS}(G))$. Since there are only a logarithmic number of layers in T_I, the size of the drawing induced by T_I is $O(\mathrm{MDS}(G) \cdot \log n)$.

Our goal is to define a decomposition tree that mimics the properties of an ideal decomposition tree. We attach to every tree node an estimator $\phi(t)$ that "behaves" like $\widetilde{V}_t$ so that we can adapt the analysis above for the drawing size that is actually computed. To be able to apply the same analysis, $\phi(t)$ should have the following properties:

1. $|cut_t| \leq \sqrt{\phi(t)}$.
2. $\phi(t)$ decreases exponentially along every path that goes down the tree.
3. $\phi(t) = O(\mathrm{MDS}(G_t))$.

Since we do not have an ideal separator procedure, we relax property 1 to $|cut_t| \leq \sqrt{\phi(t)} \cdot O(\log n)$. This degrades the approximation factor by an additional factor of $O(\log^2 n)$.

3.2. A decomposition tree with estimators.

Definition 2. *A decomposition tree T of a graph G together with a function $\phi(t)$ defined over the tree nodes is called* a decomposition tree with estimators *if the following properties are satisfied for every internal tree node $t \in T$:*

P1. $|cut_t| \leq c\sqrt{\phi(t)} \cdot \log n_t$ *for some constant c.*
P2. *For every child t' of t, $\phi(t') \leq \frac{2}{3}\phi(t)$.*
P3. $\phi(t) < 2 \cdot \mathrm{MDS}(G_t)$.

The following claim shows that a drawing induced by a decomposition tree with estimators is within $O(\log^3 n)$ factor from optimal.

Claim 6. *The size of a linear drawing of G that is induced by a decomposition tree T with estimators $\phi(\cdot)$ is $O(\mathrm{MDS}(G) \cdot \log^3 n)$.*

Proof. Consider an edge $e = (u, v)$. From Corollary 4 it follows that the number of crossings that are charged to e is bounded by $\sum_{t \in P(u,v)} |cut_t|$. Define $P(e, u)$ and $P(e, v)$ to be the paths in T from $t(e)$ to the leaves that are mapped to u and v,

respectively. It follows that

$$\sum_{t\in P(u,v)} |cut_t| \le \sum_{t\in P(e,u)} |cut_t| + \sum_{t\in P(e,v)} |cut_t|.$$

We bound each summand on the right-hand side as follows. Let $d(t,t')$ be the number of hops along the path from t to t'.

$$\begin{aligned}
\sum_{t\in P(e,u)} |cut_t| &\le \sum_{t\in P(e,u)} c\cdot\sqrt{\phi(t)}\cdot\log(n_t) && \text{(by property P1)}\\
&\le c\cdot\log n\cdot\sum_{t\in P(e,u)}\sqrt{\phi(t)}\\
&\le c\cdot\log n\cdot\sum_{t\in P(e,u)}\sqrt{\phi(t(e))\cdot\left(\tfrac{2}{3}\right)^{d(t(e),t)}} && \text{(by property P2)}\\
&= \log n\cdot O(\sqrt{\phi(t(e))})\\
&= O(\sqrt{\mathrm{MDS}(G_{t(e)})}\cdot\log n) && \text{(by property P3).}
\end{aligned}$$

By properties P1 and P3, for each tree node t, $|cut_t| = O(\sqrt{\mathrm{MDS}(G_t)}\cdot\log n)$; hence we can bound the total number of crossings charged to the edges in cut_t by $O(\log^2 n\cdot\mathrm{MDS}(G_t))$.

Fix some height in T, and consider all of the tree nodes $t_1,\ldots,t_r$ that are of this height. Notice that the sets V_{t_i} are disjoint and thus, by the superadditivity property (Proposition 1), $\mathrm{MDS}(G)\ge\sum_{i=1}^{r}\mathrm{MDS}(G_{t_i})$. It follows that the total number of crossings charged to the edges in $\cup_{i=1}^{r} cut_{t_i}$ is bounded by

$$\sum_i O(\log^2 n\cdot\mathrm{MDS}(G_{t_i}))\le O(\log^2 n\cdot\mathrm{MDS}(G)).$$

Observe that the tree T has height $O(\log n)$. This is because the estimator $\phi(\cdot)$ decreases exponentially along each path from the root to a leaf, and the estimator of the root of the tree is $O(m^2+n)$ since $\mathrm{MDS}(G)=O(m^2+n)$. It follows that the total number of crossing points is $O(\log^3 n\cdot\mathrm{MDS}(G))$. □

Remark. A better approximation algorithm for separators would yield a better approximation factor for the crossing number and the layout area. In particular, an $\alpha(n)$-approximation algorithm for separators would imply a tightened property P1, i.e., $cut_t\le c\sqrt{\phi(t)}\cdot\alpha(n_t)$, for every tree node t. This would imply an approximation factor of $O(\alpha^2(n)\cdot\log n)$ for the crossing number and $O(\alpha^2(n)\cdot\log^2 n)$ for the layout area (see section 6). This improves the corresponding results of Bhatt and Leighton (Theorems 17 and 19 in [BL84]) by a logarithmic factor and a log-square factor, respectively.

4. Constructing a decomposition tree with estimators. Our goal is to construct a decomposition tree T of G with estimators $\phi(t)$. The challenge in computing the tree is due to the requirements that (i) $\sqrt{\phi(t)}$ cannot be too small since we must be able to find a separator of size $c\sqrt{\phi(t)}\log n_t$ for some constant c; (ii) $\phi(t)$ must decay exponentially along every "downward" path; and (iii) $\phi(t)$ cannot be greater than (twice) $\mathrm{MDS}(G_t)$. The algorithm avoids an exponential search by using "failures" for pruning backtracking and for rebalancing.

Throughout the description, let $G = (V, E)$ denote the graph for which a decomposition tree is being computed. To simplify notation, we refer to induced subgraphs of G simply by their vertex sets.

Our algorithm is recursive. The input to the recursive procedures consists of the following inputs:

- a vertex subset V' (the goal is to compute a decomposition tree for the subgraph $G' = (V', E')$ induced by V');
- a "guess" g on the estimate $\phi(r')$ of the root of this tree;
- the second procedure also inputs a weight function on the vertices in V'. This weight function is either the trivial function I that assigns a unit weight to each vertex or the weight function $W_{V',B}$ with respect to the home set V' and an outside set B. To simplify notation, we sometimes omit the home set and the outside set and simply denote the nontrivial weight function by W.

Let decompose(V', g) denote our first recursive procedure. The procedure returns one of the following:

- "fail" if the guess is too small (i.e., $g < \mathrm{MDS}(V')$); or
- "success," in which case the procedure also returns a decomposition tree T' of G' with estimators $\phi(t)$. The estimator of the root r' of T' satisfies $\phi(r') \leq g$.

Since $n + \binom{m}{2}$ is an upper bound on $\mathrm{MDS}(G)$, the decomposition tree with estimators of the input graph $G = (V, E)$ is computed by calling decompose$(V, n + \binom{m}{2})$.

The procedure decompose(V', g) searches for an approximation of the smallest feasible guess in the range $[1, g]$. This search is performed by calling a second procedure test-decompose(V', g', I) for various values of g'. The output of procedure test-decompose is also either "fail" (when the guess is too small) or "success" (when the guess is large enough but might be too large). In case of success, test-decompose(V', g', I) returns a decomposition tree with "lax" estimators. The estimate $\phi(r')$ assigned to the root by the lax estimators might not satisfy property P3. That is, it outputs a success also in case the estimators $\phi(t)$ satisfy properties P1–P3 for all tree nodes but the root node, and the estimator for the root $\phi(r')$ satisfies P1 and P2 but $\phi(r') \geq 2 \cdot \mathrm{MDS}(V')$. The two procedures could be merged into a single procedure in which a range of guess values is input (the range is $[1, g]$ for decompose and $[g, g]$ for test-decompose), and a search takes place within the guess range for an approximation of the smallest feasible guess. However, the partition into two procedures seems to simplify the description.

The procedure decompose(V', g). This procedure calls successively test-decompose$(V', g/2^i, I)$ for $i = 0, 1, 2, \ldots$ until test-decompose returns "fail." We distinguish between two cases.

Case 1. The first call to test-decompose (with $i = 0$) returns "fail." In this case, decompose(V', g) returns "fail" as well.

Case 2. Some calls to test-decompose return "success." Let s be the index of the last success. In this case, decompose(V', g) returns "success" with the decomposition tree and estimators computed by test-decompose$(V', g/2^s, I)$.

The failure with $s+1$ in Case 2 is used to guarantee property P3 with respect to $\phi(r')$. In Lemma 8, we prove that if test-decompose(V', g', I) fails, then $\mathrm{MDS}(V') > g'$. Therefore, the failure with $s + 1$ implies that $g/2^s < 2 \cdot \mathrm{MDS}(V')$, and property P3 is satisfied.

The procedure test-decompose(V', g, W). This procedure has several steps.

Step 0. If V' contains a single vertex v, then return "fail" if $g < 1$ and "success"

if $g \geq 1$. In case of success, the decomposition tree is a single leaf t with $V_t = \{v\}$ and $\phi(t) = 1$. If G' contains more than one vertex, go to the next step.

Step 1. Find a simultaneous separator of V'. (Recall that such a separator is a $(\frac{1}{3}, \frac{2}{3})$-separator with respect to the weight and a $(\frac{1}{4}, \frac{3}{4})$-separator with respect to the cardinality.) Denote the two parts by V_1' and $V_2' = V' - V_1'$. Note that when the weight function is the trivial function I, a nonweighted $(\frac{1}{4}, \frac{3}{4})$-separator suffices.

Step 2. Let c_S be the constant such that by Lemma 5 there exists a simultaneous separator of V' of size $c_S \cdot \sqrt{\mathrm{MDS}(G')}$. If the size of the separator is greater than $c_S \cdot \sqrt{g} \cdot \log n'$, where $n' = |V'|$, then return "fail." Otherwise, go to the next step. The reason for the failure is that we are guaranteed to find a separator of size at most $c_S \cdot \sqrt{\mathrm{MDS}(V')} \log n'$, and thus we have a proof that $\mathrm{MDS}(V') > g$.

Step 3. Call decompose$(V_1', 2g/3)$ and decompose$(V_2', 2g/3)$.

Step 4. Distinguish between three cases depending on the outcome of Step 3.

Case 1. Both recursive calls returned a success. In this case, we have computed decomposition trees with estimators for V_1' and V_2' that satisfy properties P1–P3. We construct a decomposition tree for V' by connecting both subtrees to a root r'. Set the estimator $\phi(r')$ to be g, and return "success" with the resulting tree and estimators. Note that, in this case, $\phi(r')$ might be much larger than $\mathrm{MDS}(V')$.[3]

Case 2. Both recursive calls returned a failure. In this case, we return "fail." The failures imply that $\mathrm{MDS}(V_i') > 2g/3$ for $i = 1, 2$; hence by superadditivity $\mathrm{MDS}(V') > 4g/3$, which justifies the failure.

Case 3. One recursive call returned a success and the other a failure. Assume decompose$(V_1', 2g/3)$ succeeded and decompose$(V_2', 2g/3)$ failed. This is the most complicated case which may occur, even when g is a good guess, due to the lack of balance between $\mathrm{MDS}(V_1')$ and $\mathrm{MDS}(V_2')$. Such an imbalance can hamper satisfying P2 (i.e., the exponential decay of the estimators) along the branch from V' to V_2'. The algorithm attempts to rebalance the partitioning by recursing on V_2' and searching for a balanced cut $cut(U, V' - U)$, where $U \subseteq V_2'$. To avoid a significant increase in the cut size (compared with $cut(V_1', V_2')$), simultaneous separators are employed as follows.

Define a weight function W_2 as follows. If the procedure test-decompose was called with the trivial weight function I, then W_2 is the weight function defined with respect to the home set V_2' and the outside set V'. Otherwise, that is, if the procedure test-decompose was called with a nontrivial weight function W, then W_2 is the weight function defined with respect to the home set V_2' and the same outside set used for W. In this way, the outside set is always the first vertex set that failed in the current sequence of failures.

[3] In fact, $\phi(r')$ can be assigned a "tighter" value which could shorten the running time. Namely, set $\phi(r')$ to be the maximum of $\frac{3}{2} \cdot \phi(r_1')$, $\frac{3}{2} \cdot \phi(r_2')$, and $(\frac{|cut(V_1', V_2')|}{c_S \log n'})^2$, where r_i' is the root of the decomposition tree computed for V_i'. In this case, if $\phi(r') = (\frac{|cut(V_1', V_2')|}{c_S \log n'})^2$, then by Lemma 5 it follows that $\phi(r') \leq \mathrm{MDS}(V')$, which means that smaller guesses for V' are not needed. However, if $\phi(r') = \frac{3}{2} \cdot \phi(r_i')$, then, from superadditivity, we can only infer that $\mathrm{MDS}(V') \geq \mathrm{MDS}(V_1') + \mathrm{MDS}(V_2') \geq \frac{1}{2} \cdot (\phi(r_1') + \phi(r_2')) \geq \frac{1}{2} \cdot \frac{2}{3} \cdot \phi(r')$. Hence $\phi(r') \leq 3 \cdot \mathrm{MDS}(V')$, and one more guess of $\phi(r')/2$ would still be required for V'.

In Case 3, continue with the following substeps.

Step 4.1. Call test-decompose(V_2', g, W_2). If this call fails, then we have proof that $\mathrm{MDS}(V') \geq \mathrm{MDS}(V_2') > g$; therefore, return "fail."

Suppose that this call succeeded. Consider the recursion tree created by the call to test-decompose(V_2', g, W_2). Since this call succeeded, the two recursive calls of decompose with the parts of V_2' and guess $2g/3$ as inputs either both succeeded or at most one of them failed. In case one of the calls failed, we recursively call test-decompose on this "failed" part with a guess g. Since, by our assumption, the calling procedure test-decompose succeeded, test-decompose must succeed on this "failed" part with a guess g. Again, this implies that the two recursive calls of decompose with the subparts of this "failed" part and the guess $2g/3$ either both succeeded or at most one failed. It follows that the recursion tree has a chain of "failure" nodes, where each such failure node denotes a failure in a call to decompose with the corresponding subgraph and guess $2g/3$. We are guaranteed that the calls to decompose with the subgraphs that correspond to the siblings of these failed nodes succeeded. This sequence of "mixed" siblings must end with two siblings for which decompose succeeded on their corresponding subgraphs and $2g/3$.

Figure 1 depicts the recursion tree until two successful siblings are encountered. We use the following notation for the vertex sets inducing the subgraphs corresponding to the siblings in this sequence. Let $F_0 = V_2'$ and $S_0 = V_1'$ be the vertex sets of the first pair of mixed siblings. Suppose that the sequence of mixed pairs is of length $x \geq 1$. Then, for $1 \leq i < x$, the sets F_i (representing failure) and S_i (representing success) denote the partition of F_{i-1} computed in Step 1 of test-decompose$(F_{i-1}, g, W_{F_{i-1},V'})$. The set F_i is the one for which the call to decompose$(F_i, 2g/3)$ failed, and the set S_i is the one for which the call to decompose$(S_i, 2g/3)$ succeeded. This chain of failures ends with the vertex set F_{x-1} that is partitioned into two sets S_x and S_{x+1} for which both calls to decompose$(S_x, 2g/3)$ and decompose$(S_{x+1}, 2g/3)$ succeeded. Let $S = S_0 \cup S_1 \cup \cdots \cup S_{x-1}$.

Step 4.2. Call to decompose$(S \cup S_x, 2g/3)$ and decompose$(S \cup S_{x+1}, 2g/3)$. If both fail, then return "fail." Lemma 8 proves that these two failures imply that $\mathrm{MDS}(V') > g$.

Suppose that decompose$(S \cup S_x, 2g/3)$ succeeded. Recall the fact that decompose$(S_{x+1}, 2g/3)$ succeeded as well. Claim 7 proves that $|cut(S \cup S_x, S_{x+1})| \leq c \cdot \sqrt{g} \cdot \log n'$. Return "success" with the decomposition tree obtained by connecting the decomposition trees computed for $S \cup S_x$ and S_{x+1} with a common root r' and the estimators returned by the two recursive calls together with $\phi(r') = g$. (A tighter assignment of $\phi(r')$ is possible, as suggested in footnote 3.)

Correctness. The following claim proves that the rebalancing performed in Step 4.2 does not increase the cut by much so that property P1 holds.

CLAIM 7. *If* test-decompose(V', g, I) *reaches Step* 4.2*, then there exists a constant* c *such that*

$$|cut(S \cup S_x, S_{x+1})| \leq c \cdot \sqrt{g} \cdot \log n'.$$

Proof. We use the same notation as in the algorithm. Notice that $S \cup S_x =$

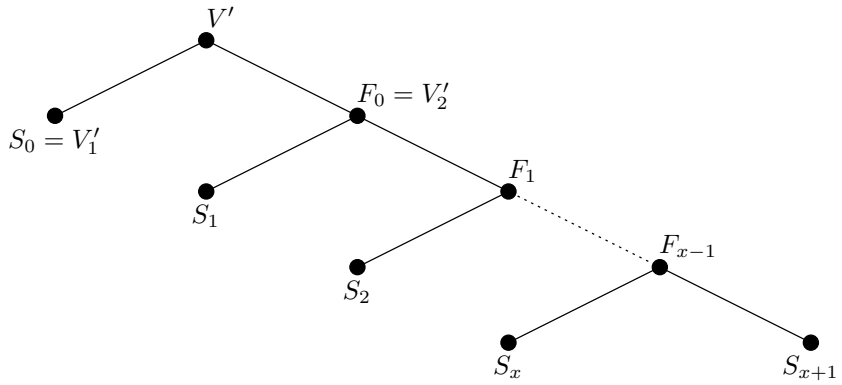


FIG. 1. *A recursion tree for which* (a) decompose$(F_i, 2g/3)$ *failed for* $i = 0, 1, \ldots, x-1$ *and* (b) decompose$(S_i, 2g/3)$ *succeeded for* $i = 0, 1, \ldots, x+1$.

$V' - S_{x+1}$, and

$$cut(S_{x+1}, V' - S_{x+1}) = cut(S_{x+1}, S_x) \cup cut(S_{x+1}, V' - F_{x-1}).$$

The success of test-decompose$(F_{x-1}, g, W_{F_{x-1},V'})$ implies that

$$|cut(S_{x+1}, S_x)| \leq c_S \cdot \sqrt{g} \cdot \log n'.$$

Since (S_{x+1}, S_x) is a $(\frac{1}{3}, \frac{2}{3})$-separator with respect to the weight function $W_{F_{x-1},V'}$ and since, for any subset $U \subseteq F_{x-1}$, the total weight $W_{F_{x-1},V'}$ over the vertices of U is $cut(U, V' - F_{x-1})$, we get

$$|cut(S_{x+1}, V' - F_{x-1})| \leq \frac{2}{3}|cut(F_{x-1}, V' - F_{x-1})|.$$

Continuing in the same manner, we get that, for $i = x-1, \ldots, 1$,

$$|cut(F_i, V' - F_i)| \leq |cut(F_i, S_i)| + \frac{2}{3}|cut(F_{i-1}, V' - F_{i-1})|.$$

For $i = 0, \ldots, x-1$, $|cut(F_i, S_i)| \leq c_S \cdot \sqrt{g} \cdot \log n'$. If we solve this recurrence relation, we get

$$|cut(S_{x+1}, V' - S_{x+1})| \leq \frac{1}{1 - \frac{2}{3}} \cdot c_S \cdot \sqrt{g} \cdot \log n'.$$

Setting $c = 3 \cdot c_S$ proves the claim. □

We now prove that failure of test-decompose(V', g, W) implies that $\mathrm{MDS}(V') > g$. This claim holds regardless of whether W is a trivial or nontrivial weight function. Notice that the guarantee is one-sided; a success might be obtained even if g is very small. (For example, consider a balanced binary tree as an input graph. All cuts have size 1, so g can be sublinear, i.e., $(\frac{3}{2})^{\log_2 n}$, but the drawing size is linear.)

LEMMA 8. *If the procedure* test-decompose(V', g, W) *fails, then* $g < \mathrm{MDS}(V')$.

Proof. The proof is by induction on $|V'|$. The induction basis is trivial (see Step 0). In the induction step, we consider the steps in which a failure can be decided upon. Assume that the lemma holds for all graphs with less than $|V'|$ vertices, and assume that $g \geq \mathrm{MDS}(V')$ and test-decompose(V', g, W) fails.

1. A failure in Step 2 contradicts the assumption because of the guarantee of the partitioning algorithm.
2. A failure in Case 2 in Step 4 contradicts the assumption as follows. By the induction hypothesis, failure of test-decompose$(V_i', 2g/3, I)$ implies $\mathrm{MDS}(V_i') > 2g/3$ for $i = 1, 2$. From the superadditivity property (Proposition 1) it follows that $\mathrm{MDS}(V') > 4g/3$, which is a contradiction.
3. A failure in Step 4.1 implies, by the induction hypothesis, that $\mathrm{MDS}(V_2') > g$. Since $\mathrm{MDS}(V') \geq \mathrm{MDS}(V_2')$, we reach a contradiction.
4. The harder case to prove is when a failure is decided upon in Step 4.2. Consider an optimal drawing D' of G', the subgraph of G induced by V'. Every point in D' is either a crossing point between two edges of G' or a vertex in V'. Let D_{x-1} denote the restriction of D' to a drawing of F_{x-1}. Recall that decompose$(F_{x-1}, 2g/3)$ failed. Therefore, by the induction hypothesis, $\mathrm{MDS}(F_{x-1}) > 2g/3$. Since D_{x-1} is a drawing of the subgraph induced by F_{x-1}, it follows that D_{x-1} contains more than $2g/3$ points.
   The points in D_{x-1} can be partitioned as follows: (a) points of the restriction of D_{x-1} to S_x, namely, vertices of S_x and crossings between edges of the subgraph induced by S_x; (b) points of the restriction of D to S_{x+1}; and (c) other crossings. Let P denote the points in D_{x-1} of type (b) or (c). By swapping S_x and S_{x+1} if needed, the number of points in P is at least half the number of points in D_{x-1} and hence greater than $g/3$.
   Let $D_{0..x}$ denote the restriction of D' to a drawing of $S \cup S_x$. The points in the drawing $D_{0..x}$ are contained in the points of the drawing D' that are not points in P. Since the number of points in P is more than $g/3$, we get that the number of points in $D_{0..x}$ is strictly less than $\mathrm{MDS}(G') - g/3$. By our assumption $g \geq \mathrm{MDS}(G')$, and thus the number of points in $D_{0..x}$ is strictly less than $2g/3$. Since $D_{0..x}$ is a drawing of $S \cup S_x$, $\mathrm{MDS}(S \cup S_x) < 2g/3$, and, by the induction hypothesis, decompose$(S \cup S_x, 2g/3, I)$ succeeds. Hence failure is not decided upon in Step 4.2, which is a contradiction. ∎

The following claim shows that the computed decomposition tree with estimators satisfies properties P1–P3.

LEMMA 9. *Let T' denote the decomposition tree with estimators $\phi(t)$ returned by a successful call to* decompose(G', g). *Then T and $\phi(t)$ satisfy properties* P1–P3 *with respect to G'.*

Proof. We extend the claim to decomposition trees computed by successful calls to test-decompose(G', g, W) with the relaxation that property P3 is not required for the root of T'.

The proof is by induction on $|V|$. It is obvious to show that the induction basis holds. The induction step proceeds as follows:

1. Property P1 is satisfied because cut_t is either a cut computed by the partitioning procedure (in which case the size of the cut is checked in Step 2) or a cut obtained by rebalancing in Step 4.2 (in which case Claim 7 guarantees that $|cut_t|$ satisfies property P1).
2. Property P2 is satisfied because, by the construction, if t' is a child of t, then $\phi(t') \leq \frac{2}{3}\phi(t)$.
3. Property P3 is satisfied by the induction hypothesis for all internal nodes of T. It holds for the root r' of T' since a success of decompose(G', g) implies a failure of test-decompose$(G', \phi(r')/2, I)$. By Lemma 8, this implies that $g/2 < \mathrm{MDS}(G')$, and property P3 follows. ∎

Polynomial running time. Let $F(n,g)$ and $G(n,g)$ denote the running times of decompose and test-decompose when the subgraph has n nodes and the guess equals g. Note that the weight function does not affect the running time. Let $p(n)$ denote a polynomial bounding the running time of the partitioning algorithm and the lines in test-decompose that do not call decompose and test-decompose. The functions $F(n,g)$ and $G(n,g)$ satisfy the following recurrences: (We make a relaxed upper bound in assuming that the functions $F(\cdot)$ and $G(\cdot)$ are monotone in both parameters.)

$$F(n,g) \leq \sum_{i=0}^{\lceil \log_2 g \rceil} G(n, g/2^i)$$

$$G(n,g) \leq p(n) + 2F\left(\frac{3}{4}n, \frac{2}{3}g\right) + G\left(\frac{3}{4}n, g\right) + 2F\left(n, \frac{2}{3}g\right).$$

The following claim proves that the running time of the decomposition algorithm is polynomial. Recall that g is always polynomial in n.

LEMMA 10. *There exist constants τ, γ, δ such that $F(n,g) \leq \tau \cdot p(n) \cdot n^{\gamma} \cdot g^{\delta}$.*

Proof idea. One may simply prove this by induction. The reason the proof works is that at each recursion step there are either (i) a constant number of recursive calls in which one of the parameters (the number of vertices or the guess value) decreases by a constant factor or (ii) a logarithmic number of recursive calls in which the guess parameters form a geometric sequence. For this reason, it was imperative that we went down a path with a geometrically decreasing number of vertices. It was also important that, in Step 4.2, once the chain of failures ends with two successful siblings, we were able to argue that the guess value should decrease by a constant factor.

Claim 6 and Lemmas 9 and 10 imply the following theorem. Note that the algorithm computes a linear drawing.

THEOREM 11. *There exists an $O(\log^3 n)$-approximation algorithm for the minimum drawing size of bounded degree graphs.*

Coping with skewed weight functions. As mentioned above, for some weight functions $W_{A,B}$, there may be a vertex v such that $W_{A,B}(v) > \frac{2}{3}\sum_{v \in A} W_{A,B}(v)$. We call such weight functions *skewed* weight functions. In the case of skewed weight functions, we cannot apply Lemma 5 to find a simultaneous separator. Below, we show how to handle such cases. First, we show that in such cases the total weight is bounded and then prove that, because of the bounded weight, any cut that is a $(\frac{1}{4}, \frac{3}{4})$-separator with respect to the number of vertices suffices.

CLAIM 12. *Let $G = (V,E)$ denote a graph with maximum degree Δ. For any two subsets of vertices $A, B \subseteq V$ and a vertex $v \in A$ such that $W_{A,B}(v) > \frac{2}{3}\sum_{v \in A} W_{A,B}(v)$, $|cut(A, B - A)| \leq \frac{3}{2} \cdot \Delta$.*

Proof. Recall that $\sum_{v \in A} W_{A,B}(v) = |cut(A, B - A)|$. The claim follows since by definition $W_{A,B}(v) \leq \Delta$. □

Simultaneous separators are computed in Steps 1 and 4.1 of test-decompose. Whenever the weight function is skewed and nontrivial, a simultaneous separator cannot be computed. Instead, a $(\frac{1}{4}, \frac{3}{4})$-separator with respect to the number of vertices is computed in such cases. To maintain the validity of the algorithm, we need to revise the proof of Lemma 7.

Proof of Lemma 7 (revised). We use the same notation as in the algorithm. Notice that $S \cup S_x = V' - S_{x+1}$, and

$$cut(S_{x+1}, V' - S_{x+1}) = cut(S_{x+1}, S_x) \cup cut(S_{x+1}, V' - F_{x-1}).$$

The success of test-decompose$(F_{x-1}, g, W_{F_{x-1},V'})$ implies that

$$|cut(S_{x+1}, S_x)| \leq c_S \cdot \sqrt{g} \cdot \log n'.$$

If $cut(S_{x+1}, S_x)$ is a simultaneous separator, then, since (S_{x+1}, S_x) is a $(\frac{1}{3}, \frac{2}{3})$-separator with respect to the weight function $W_{F_{x-1},V'}$ and since, for any subset $U \subseteq F_{x-1}$, the total weight $W_{F_{x-1},V'}$ over the vertices of U is $cut(U, V' - F_{x-1})$, we get

$$|cut(S_{x+1}, V' - F_{x-1})| \leq \frac{2}{3}|cut(F_{x-1}, V' - F_{x-1})|.$$

Suppose that $cut(S_{x+1}, S_x)$ is not a simultaneous separator but a $(\frac{1}{4}, \frac{3}{4})$-separator with respect to the number of vertices. This happens when the weight function $W_{F_{x-1},V'}$ is skewed. By Claim 12, we have in this case $|cut(S_{x+1}, V' - F_{x-1})| \leq \frac{3}{2}\Delta$ (where Δ is the degree bound).

Combining both cases, we have

$$\begin{aligned} |cut(S_{x+1}, V' - F_{x-1})| &\leq \max\left\{\frac{2}{3}|cut(F_{x-1}, V' - F_{x-1})|, \frac{3}{2}\Delta\right\} \\ &\leq \frac{2}{3}|cut(F_{x-1}, V' - F_{x-1})| + \frac{3}{2}\Delta. \end{aligned}$$

Continuing in the same manner, we get that, for $i = x - 1, \ldots, 1$,

$$|cut(F_i, V' - F_i)| \leq |cut(F_i, S_i)| + \frac{2}{3}|cut(F_{i-1}, V' - F_{i-1})| + \frac{3}{2}\Delta.$$

For $i = 0, \ldots, x - 1$, $|cut(F_i, S_i)| \leq c_S \cdot \sqrt{g} \cdot \log n'$. Solving this recurrence relation, we get

$$|cut(S_{x+1}, V' - S_{x+1})| \leq \frac{1}{1 - \frac{2}{3}}\left(c_S \cdot \sqrt{g} \cdot \log n' + \frac{3}{2}\Delta\right).$$

Setting $c = 3c_S + \frac{9}{2}\Delta$ proves the claim. ∎

5. Approximating $\sqrt{2}$-bifurcators. An (F, α)-bifurcator is a decomposition tree in which $|cut_t| \leq F \cdot \alpha^{-depth(t)}$ for every tree node t. An optimal α-bifurcator is an (F, α)-bifurcator with the minimum F. In our algorithm, we require that the bounds on the cut sizes decrease geometrically by $\frac{2}{3}$ relative to *every* node and not only the root. This property was required so that we could bound the sum of the cuts along a path from an internal tree node t to a leaf by $O(\phi(t))$. The following corollary is shown in [BL84, LR99].

COROLLARY 13. *Given an $O(\log^k n)$-approximation for the size of drawing on the plane, there exists an $O(\log^{(k+1)/2} n)$-approximation algorithm for finding the optimal $\sqrt{2}$-bifurcator.*

Theorem 11 implies an improvement of the approximation factor of $\sqrt{2}$-bifurcators by a $\sqrt{\log n}$ factor to $O(\log^2 n)$. As mentioned in the introduction, Bhatt and Leighton [BL84] demonstrated that $\sqrt{2}$-bifurcators are useful for minimizing capacitive delay, producing fault tolerant layouts (layouts using prefabricated chip and regular layouts), minimizing wire crossing in layouts, and more.

6. VLSI layouts. In this section, we present a VLSI layout algorithm for graphs of maximum degree four. The algorithm embeds a graph G in a square host grid of size $O(\text{AREA}(G) \cdot \log^4 n)$.

An $O(\log^5 n)$-approximate VLSI layout area can be derived by applying planar graph embedding algorithms on the drawing computed in section 4, the size of which is $O(\text{MDS} \cdot \log^3 n)$ [L80, V81]. This is an improvement by a logarithmic factor over the approximation factor of the VLSI layout algorithm of Bhatt and Leighton [BL84]. We show how an additional logarithmic factor can be saved.

6.1. Weighted version of Leiserson's algorithm. Leiserson's algorithm for computing layouts of graphs is based on the graphs satisfying a separator theorem. The separator theorem is stated in terms of the *number* of vertices. Namely, every n-vertex graph in a given family can be separated into two disjoint graphs, each containing at least a constant fraction of the vertices, by removing at most $f(n)$ edges. For example, in bounded degree planar graphs, the separator is guaranteed to be bounded by the square-root of the number of vertices, and the size of each part is bounded by $\frac{2}{3}$ of the original number of vertices. Planar graph embedding algorithms require $O(n \cdot \log^2 n)$ area.

We consider a "weighted" version of Leiserson's algorithm in which subgraphs are assigned weights. The input to the weighted version of Leiserson's algorithm consists of a graph and a decomposition tree with weights $Wt(t)$ defined over the tree nodes. Loosely, the cut size in every internal tree node is bounded by the square-root of the weight of the tree node, and the weight of each node is divided roughly equally between its two children. The weighted version of Leiserson's algorithm returns a VLSI layout of area $O(Wt(r) \cdot \log^2 Wt(r))$, where r denotes the root of T. If $Wt(r) = O(\text{MDS}(G) \cdot \log^2 n)$, then $\log Wt(r) = O(\log n)$, and, by Proposition 2, an approximation factor of $O(\log^4 n)$ follows for the VLSI layout.

The following four properties of the decomposition tree T and the tree node weights $Wt(t)$ are sufficient for the "success" of the weighted version of Leiserson's algorithm:

Nontriviality. For each leaf ℓ, $Wt(\ell) \geq 1$.

Superadditivity. For any two siblings t_ℓ and t_r in the tree, with a parent t,

$$Wt(t_\ell) + Wt(t_r) \leq Wt(t).$$

Balance. For any two siblings t_ℓ and t_r in the tree,

$$\max\{Wt(t_\ell), Wt(t_r)\} \leq 2 \cdot \min\{Wt(t_\ell), Wt(t_r)\}.$$

(Note that the above two conditions imply that both the weights $Wt(t_\ell)$ and $Wt(t_r)$ are at most $\frac{2}{3} Wt(t)$.)

Separator. For any internal tree node t, $|cut_t| = O(\sqrt{Wt(t)})$.

We outline the weighted version of Leiserson's algorithm. First, the algorithm defines a function $A(t)$ over the tree nodes that specifies the area allocated for embedding G_t. The function $A(t)$ is defined as follows:

$$A(t) = \begin{cases} 1 & \text{if } t \text{ is a leaf,} \\ \left(\sqrt{A(t_\ell) + A(t_r)} + \alpha \cdot |cut_t|\right)^2 & \text{otherwise,} \end{cases}$$

where t_ℓ and t_r denote the children of t and $\alpha = 2\sqrt{3}$.

Given a rectangle R_t with aspect ratio at least $\frac{1}{3}$ and area $A(t)$, the algorithm computes an embedding of G_t in R_t as follows. A subrectangle R'_t of area $A(t_\ell)+A(t_r)$ that is similar to R_t is allocated for embedding G_ℓ and G_r. The subrectangle R'_t is divided between G_{t_ℓ} and G_{t_r} in proportion to the weights $Wt(t_\ell)$ and $Wt(t_r)$. After G_{t_ℓ} and G_{t_r} are embedded in the subrectangles allocated for them, the edges of cut_t are embedded using the unused rows and columns of R_t. (At most two columns and two rows are used for embedding each edge in the cut.) In Claim 14, it is proved that there is a sufficient number of unused rows and columns left in R_t for routing the edges of cut_t.

We briefly point out why each of the properties is required: Nontriviality and superadditivity guarantee that the size of a rectangle allocated for a subgraph is not smaller than the size of the subgraph. Superadditivity also makes sure that the subrectangle allocated for the children of t is not larger than the rectangle allocated for t. The balance property makes sure that the division of the rectangle does not create subrectangles whose aspect ratios are below $\frac{1}{3}$. Hence, throughout the algorithm, the aspect ratios of the rectangles that serve as hosts for subgraphs are at least $\frac{1}{3}$. This also implies that the host grid for the whole graph can be a rectangle of aspect ratio $\frac{1}{3}$ rather than a square. The separator property is used to solve the recurrence and prove the bound on the required area.

The following claim shows that the weighted version of Leiserson's algorithm succeeds in embedding a graph of maximum degree four in a rectangle of area $A(t)$.

CLAIM 14. *Let T denote a decomposition tree with node weights $Wt(t)$ of a graph G of maximum degree four. Suppose that the node weights satisfy the nontriviality, superadditivity, and balance properties. If R is a rectangle whose aspect ratio is at least $\frac{1}{3}$ and whose area is at least $A(t)$, then the weighted version of Leiserson's algorithm succeeds in embedding G in R.*

Proof. The proof follows [L80, Thm. 6]. In particular, we rely on two observations in Leiserson's proof. One observation is that aspect ratios do not deteriorate. Namely, suppose we divide a rectangle R whose aspect ratio is at least $\frac{1}{3}$ into two subrectangles R_1 and R_2 by cutting it along the longer side. Suppose also that this cut is made between a third and two thirds of the side's length so that the area of the larger subrectangle is at most twice the area of the smaller subrectangle. Then the aspect ratios of R_1 and R_2 are also at least $\frac{1}{3}$. The second observation is that a new edge can be routed in a given embedding by introducing at most two rows and two columns. (This is called "slicing" in [L80].)

The proof is by induction on the height of the tree node. The induction basis obviously holds for leaves. Let R'_t denote the subrectangle of R_t allocated for G_{t_ℓ} and G_{t_r}. By definition, R'_t and R_t have the same aspect ratio. The area of R'_t equals $A(t_\ell) + A(t_\ell)$. (There is a "rounding" issue that we ignore to make the presentation simpler.) We need to show that there are at least $2 \cdot |cut_t|$ spare rows and columns in R_t for routing the edges of cut_t.

Let $L(R)$ and $W(R)$ denote the length and width of a rectangle R. (Assume $W(R_t) \leq L(R_t)$.) Denote the aspect ratio of a rectangle R by $\sigma(R)$, namely, $\sigma(R) = W(R)/L(R)$. The width of R_t satisfies

$$W(R_t) = \frac{area(R_t)}{L(R_t)} = \frac{area(R_t) \cdot \sigma(R_t)}{W(R_t)}.$$

Since $area(R_t) \geq A(t)$,

$$W(R_t) \geq \sqrt{A(t) \cdot \sigma(R_t)}.$$

Note that, since R_t and R'_t are similar, they have the same aspect ratio; hence

$$W(R'_t) = \sqrt{A(t_\ell) + A(t_r)}\sqrt{\sigma(R_t)}.$$

By the last two equations and the definition of $A(t)$, it follows that

$$\begin{aligned} W(R_t) &\geq \sqrt{\sigma(R_t) \cdot A(t)} \\ &= \sqrt{\sigma(R_t)} \cdot \left(\sqrt{A(t_\ell) + A(t_r)} + 2\sqrt{3} \cdot |cut_t| \right) \\ &= W(R'_t) + \sqrt{\sigma(R_t)} \cdot 2\sqrt{3} \cdot |cut_t|. \end{aligned}$$

Since $\sigma(R_t) \geq \frac{1}{3}$, it follows that $W(R_t) - W(R'_t) \geq 2 \cdot |cut_t|$. Hence there are enough spare rows. Since $L(R_t) - L(R'_t) = (W(R_t) - W(R'_t))/\sigma(R_t)$, there are also enough spare columns, and the claim follows. □

The following claim shows that if the decomposition tree satisfies all four properties, then $A(r) = O(Wt(r) \cdot \log^2 Wt(r))$, where r denotes the root of the decomposition tree.

Claim 15. *Let T denote a decomposition tree with node weights $Wt(t)$ of a graph G of maximum degree four. Suppose that the node weights satisfy the nontriviality, superadditivity, balance, and separator properties. There exists a constant c_A such that*

$$A(t) = c_A \cdot Wt(r) \cdot \log^2 Wt(r).$$

Proof. The proof follows [L80, section 5]. Recall that $\alpha = 2\sqrt{3}$, and let $\beta = \max_{t \in T}\{|cut_t|/\sqrt{Wt(t)}\}$. The separator property implies that β is a constant. Define the function $B(t)$ over the tree nodes as follows:

$$B(t) = \begin{cases} 1 & \text{if } t \text{ is a leaf,} \\ \max\{B(t_\ell), B(t_r)\} + \alpha \cdot \beta & \text{otherwise.} \end{cases}$$

We prove by induction that $A(t) \leq B^2(t) \cdot Wt(t)$. This obviously holds for leaves. The induction step is proved as follows:

$$\begin{aligned} \sqrt{A(t)} &= \sqrt{A(t_\ell) + A(t_r)} + \alpha \cdot |cut_t| \\ &\leq \sqrt{B^2(t_\ell) \cdot Wt(t_\ell) + B^2(t_r) \cdot Wt(t_r)} + \alpha\beta\sqrt{Wt(t)} \\ &\leq \max\{B(t_\ell), B(t_r)\} \cdot \sqrt{Wt(t)} + \alpha\beta\sqrt{Wt(t)} \\ &= B(t) \cdot \sqrt{Wt(t)}. \end{aligned}$$

The second inequality follows from the induction hypothesis and the definition of β. The third inequality follows from the superadditivity property.

The balance and superadditivity properties imply that the depth of T is at most $\log_{2/3} Wt(t)$, and hence it follows that $B(t) \leq \alpha\beta \log_{2/3} Wt(t)$, and the claim follows. □

Remark. If the balance property is relaxed so that $\max\{Wt(t_\ell), Wt(t_r)\} \leq (1 - \sigma) \cdot Wt(t)$, then the aspect ratios of the rectangles allocated in the algorithm are lower bounded by σ. The proof of Claim 14 required that $\alpha \geq 2/\sqrt{\sigma}$. In the proof of Claim 15, $A(t)$ is proportional to α^2. Therefore, the balance property can be relaxed at the cost of increasing $A(t)$.

6.2. Computing a weighted decomposition tree. Given a graph G, we show how to construct a decomposition tree T and a weight function $Wt(\cdot)$ that obeys the properties sufficient for the success of the weighted version of Leiserson's algorithm. Observe that recursively separating an optimal drawing of G generates a decomposition tree that satisfies these properties if the weight function equals the size of the subdrawing. We show how to compute an "approximation" of such a tree in which the weight of the root is $O(\mathrm{MDS}(G)\log^2 n)$.

At first glance, it is tempting to use the decomposition tree computed for the minimal drawing as the decomposition tree required here and to use the estimator function computed before as the required weight function. Two obstacles obstruct this approach: (a) The decomposition tree is not balanced since the ratio of estimators of two siblings is not bounded. (b) The decomposition tree may not obey the superadditivity property, as we may have two siblings with estimators that add up to more than the estimator of the parent (i.e., the estimators of each of the children of an internal node t can be $2\phi(t)/3$). Violating balance is solved by rebalancing, but violating superadditivity makes $\phi(t)$ a bad candidate for a weight function. Superadditivity is obtained by presenting a weight function that is computed bottom-up (to ensure superadditivity). Balance is obtained afterward by rebalancing the tree with respect to the weights in a top-down fashion.

6.2.1. The new weight function. Let T denote the decomposition tree of G with estimator $\phi(t)$ computed in section 4. Let $CW(t)$ be the maximum sum of cut sizes along a path from t to a leaf in the subtree rooted at t, where the maximum is taken over all such paths.

Definition 3. *The weight function $Wt(t)$ is defined as follows:*

$$Wt(t) = \begin{cases} 1 & \text{if } t \text{ is a leaf,} \\ \max\{\, Wt(t_\ell) + Wt(t_r), CW^2(t)\} & \text{otherwise.} \end{cases}$$

The following lemma bounds the weight of the root of T.

Lemma 16. *For some constant c_W, $Wt(t) \le c_W \cdot \mathrm{MDS}(G_t) \cdot \log^2(n_t + 1)$.*

Proof. The proof is by induction. The induction basis for the leaves is obvious. Consider an internal tree node t. If $Wt(t) = Wt(t_\ell) + Wt(t_r)$, then the claim follows by the superadditivity of the minimum drawing size.

If $Wt(t) = CW^2(t)$, note that the upper bounds on the sizes of the cuts along any path from t to any of its leaves decay exponentially and that the size of the first cut (and hence the sum of all of the cuts along a path from t to a leaf) is $O(\sqrt{\phi(t)}\log n_t) = O(\sqrt{\mathrm{MDS}(G_t)}\log n_t)$. The square of the sum of these cuts is $O(\mathrm{MDS}(G_t)\log^2 n_t)$. For an appropriately chosen c_W, the claim follows. ∎

The weight function $Wt(t)$ satisfies all of the properties required except for the balance property. We show how to achieve balance in the following subsection.

6.2.2. Rebalancing the tree. Rebalancing proceeds in two steps: First, the tree is rebalanced so that the weight of each node is at most two-thirds the weight of its parent. Second, weights are inflated, if necessary, so that balance is achieved.

Procedure rebalance($\mathcal{T}$). The rebalance($\mathcal{T}$) procedure balances the tree recursively and assigns weights $Wt_2(\cdot)$ to tree nodes. The weight of each node is at most two-thirds the weight of its parent. The input consists of a collection of trees (inclusive of the case of a single tree) denoted by $\mathcal{T}$, where the initial input consists of a single tree. The procedure outputs a decomposition tree over the union of the leaves of trees

in $\mathcal{T}$. The following notation is used: the weight of a tree, $Wt(T')$, is defined to be the weight of its root, $Wt(r')$. Given a collection $\mathcal{T}$ of trees, let $Wt(\mathcal{T}) = \sum_{T' \in \mathcal{T}} Wt(T')$.

The procedure rebalance($\mathcal{T}$). Assume $\mathcal{T} = \{T_1, \ldots, T_p\}$.

1. If there exists any j such that $\sum_{i=1}^{j} Wt(T_i)$ is between $\frac{1}{3}$ and $\frac{2}{3}$ of $Wt(\mathcal{T})$, we create the two collections $\mathcal{T}_1 = \{T_1, \ldots, T_j\}$ and $\mathcal{T}_2 = \{T_{j+1}, \ldots, T_k\}$. The separating cut is of size 0.
2. If there is no such j, then there exists a j such that $\sum_{i=1}^{j-1} Wt(T_i)$ and $\sum_{i=j+1}^{p} Wt(T_i)$ are both less than $\frac{1}{3} Wt(\mathcal{T})$. Starting at the root $r(T_j)$ of T_j, walk down the tree, following the heavier subtree until the first subtree T' whose weight is less than $\frac{2}{3} Wt(T_j)$ is reached. Let the subtrees hanging from the path from $r(T_j)$ to T' (other than T') be $T'_1, \ldots, T'_q$ (where T'_q denotes the sibling of T'). By Lemma 17, $\sum_{i=1}^{q} Wt(T'_i) < \frac{2}{3} Wt(T_j)$.
3. We have four collections:
$$\mathcal{T}_a = \{T_1, \ldots, T_{j-1}\}, \qquad \mathcal{T}_b = \{T_{j+1}, \ldots, T_p\},$$
$$\mathcal{T}_c = \{T'\}, \qquad \mathcal{T}_d = \{T'_1, \ldots, T'_q\}.$$
Let $MIN(a,b) \in \{a,b\}$ be the index of the collection with the smaller weight between collections $\mathcal{T}_a$ and $\mathcal{T}_b$. Recall that the weight of a collection $\mathcal{T}$ is the sum of the weights of the trees in $\mathcal{T}$. Similarly, $MAX(a,b) \in \{a,b\}$ is defined to be the index of the larger weighted collection. Define $MIN(c,d)$ and $MAX(c,d)$ similarly. Let
$$\mathcal{T}_1 = \mathcal{T}_{MIN(a,b)} \cup \mathcal{T}_{MAX(c,d)},$$
$$\mathcal{T}_2 = \mathcal{T}_{MAX(a,b)} \cup \mathcal{T}_{MIN(c,d)}.$$
4. Call rebalance($\mathcal{T}_1$) and rebalance($\mathcal{T}_2$), and let S_i denote the decomposition tree computed for $\mathcal{T}_i$. Let S be a decomposition tree for $\mathcal{T}$ obtained by connecting S_1 and S_2 to a root s. Set $Wt_2(s) = Wt(\mathcal{T})$.

Note that the weight function $Wt_2(\cdot)$ satisfies all of the properties except for balance. In particular: (a) The separator property holds since the size of a cut is either zero or is bounded by $CW(r(T_j))$, where $r(T_j)$ is the root of T_j, which is bounded by $\sqrt{Wt(T_j)}$. (b) The superadditivity property holds since either $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$ or T_j is partitioned into the collections $\mathcal{T}_c$ and $\mathcal{T}_d$, and $Wt(T_j) \geq Wt(\mathcal{T}_c) + Wt(\mathcal{T}_d)$. Note also that if a single tree is rebalanced, then the weight of the root after rebalancing equals the weight of the root before balancing. So Lemma 16 implies that the weight of the root s of the decomposition tree computed by rebalance($\{T\}$) satisfies

$$Wt_2(s) \leq c_W \cdot \mathrm{MDS}(G_t) \cdot \log^2(n+1).$$

The following lemmas prove that the weight of every node is at most two-thirds the weight of its parent.

LEMMA 17. *In step* 2 *above,* $\sum_{i=1}^{q} Wt(T'_i) < \frac{2}{3} Wt(T_j)$.

Proof. Consider two cases: (a) If $Wt(T') > Wt(T_j)/3$, then it follows by superadditivity. (b) If $Wt(T') \leq Wt(T_j)/3$, then $Wt(T'_q) \leq Wt(T_j)/3$ also. Since the weight of the parent of T' and T'_q is not less than $\frac{2}{3} Wt(T_j)$, we have $\sum_{i=1}^{q-1} Wt(T'_i) < Wt(T_j)/3$, and the claim follows. ∎

LEMMA 18. *In the decomposition tree returned by* rebalance($\mathcal{T}$)*, the weight of a nonroot node is at most two-thirds the weight of its parent.*

Proof. Since $Wt(\mathcal{T}_{MIN(a,b)}) \leq \frac{1}{2}(Wt(\mathcal{T}) - Wt(T_j))$ along with $Wt(\mathcal{T}_{MAX(c,d)}) < \frac{2}{3} Wt(T_j)$, we have $Wt(\mathcal{T}_1) < \frac{2}{3} Wt(\mathcal{T})$. For the bound on $Wt(\mathcal{T}_2)$, we distinguish two cases. (a) $Wt(T_j) \leq \frac{2}{3} Wt(\mathcal{T})$. In this case, since $Wt(\mathcal{T}_{MIN(c,d)}) \leq \frac{1}{2} Wt(T_j)$ and $Wt(\mathcal{T}_{MAX(a,b)}) < \frac{1}{3} Wt(\mathcal{T})$, we have $Wt(\mathcal{T}_2) < \frac{2}{3} Wt(\mathcal{T})$. (b) $Wt(T_j) > \frac{2}{3} Wt(\mathcal{T})$.

In this case, since $Wt(\mathcal{T}_{MAX(a,b)}) \leq Wt(\mathcal{T}) - Wt(T_j)$, we have $Wt(\mathcal{T}_2) \leq Wt(\mathcal{T}) - Wt(T_j) + \frac{1}{2} Wt(T_j) < \frac{2}{3} Wt(\mathcal{T})$. □

Inflating weights and achieving balance. The balance property is obtained by inflating weights using the following local transformation. For every nonroot internal tree node t, let t' denote its sibling. Define $Wt_3(t)$ as follows:

$$Wt_3(t) = \max\left\{ Wt_2(t), \frac{1}{2} Wt_2(t') \right\}.$$

After the inflation, all of the other properties are preserved. For example, the superadditivity property is preserved since, if $Wt_3(t_\ell) > Wt_2(t_\ell)$, then $Wt_3(t_\ell) + Wt_3(t_r) \leq \frac{3}{2} Wt_2(t_r) \leq \frac{3}{2} \cdot \frac{2}{3} Wt_2(t) = Wt_2(t) \leq Wt_3(t)$, where t denotes the parent of t_ℓ and t_r.

Having computed a decomposition tree with weights that satisfy all of the properties, we conclude with the following theorem.

THEOREM 19. *Every graph G of maximum degree four can be embedded in polynomial time in any grid of aspect ratio at least $\frac{1}{3}$ and area $O(\text{AREA}(G) \log^4 n)$.*

Acknowledgments. The authors would like to thank Seffi Naor, Chandan Deep Nath, János Pach, Satish Rao, and Farhad Shahrokhi for discussions about these problems. The authors wish to thank the anonymous reviewers for many useful comments and suggestions.

REFERENCES

[BL84] S. N. BHATT AND F. T. LEIGHTON, *A framework for solving VLSI graph layout problems*, J. Comput. System Sci., 28 (1984), pp. 300–343.

[ENRS99] G. EVEN, J. NAOR, S. RAO, AND B. SCHIEBER, *Fast approximate graph partitioning algorithms*, SIAM J. Comput., 28 (1999), pp. 2187–2214.

[G00] S. GUHA, *Nested graph dissection and approximation algorithms*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, Redondo Beach, CA, 2000, pp. 126–135.

[GJ79] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[GJ83] M. R. GAREY AND D. S. JOHNSON, *Crossing number is NP-complete*, SIAM J. Algebraic Discrete Methods, 4 (1983), pp. 312–316.

[GM90] H. GAZIT AND G. L. MILLER, *Planar separators and the Euclidean norm*, in Proceedings of the 1st International Symposium on Algorithms (SIGAL), Lecture Notes in Comput. Sci. 450, Springer-Verlag, Heidelberg, 1990, pp. 338–347.

[L80] C. E. LEISERSON, *Area-efficient graph layouts (for VLSI)*, in Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science, Syracuse, NY, 1980, pp. 270–281.

[L83] F. T. LEIGHTON, *Complexity Issues in VLSI*, MIT Press, Cambridge, MA, 1983.

[LR99] T. LEIGHTON AND S. RAO, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, J. ACM, 46 (1999), pp. 787–832.

[LT79] R. J. LIPTON AND R. E. TARJAN, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.

[PSS96] J. PACH, F. SHAHROKHI, AND M. SZEGEDY, *Applications of the crossing number*, Algorithmica, 16 (1996), pp. 111–117.

[SS00] F. SHAHROKHI, AND W. SHI, *On crossing sets, disjoint sets, and pagenumber*, J. Algorithms, 34 (2000), pp. 40–53.

[SSSV97] F. SHAHROKHI, O. SÝKORA, L. A. SZÉKELY, AND I. VRŤO, *Crossing numbers: Bounds and applications*, in Intuitive Geometry, Bolyai Soc. Math. Stud. 6, János Bolyai Math. Soc., Budapest, 1997, pp. 179–206.

[U84] J. D. ULLMAN, *Computational Aspects of VLSI*, Computer Science Press, Rockville, MD, 1984.

[V81] L. G. VALIANT, *Universality considerations in VLSI circuits*, IEEE Trans. Comput., 30 (1981), pp. 135–140.

SIAM J. COMPUT.
Vol. 32, No. 1, pp. 253–280

COMPUTATIONAL COMPLEXITY OF COMPACTION TO REFLEXIVE CYCLES*

NARAYAN VIKAS†

Abstract. In this paper, we solve a widely publicized open problem posed by Peter Winkler in 1988. The problem is to decide whether or not it is possible to partition the vertices of a graph into four distinct nonempty sets A, B, C, and D, such that there is no edge between the sets A and C, and between the sets B and D, and that there is at least one edge between any other pair of distinct sets. Winkler asked whether this problem is NP-complete. We show in this paper that it is NP-complete. We study the problem as the compaction problem for a reflexive 4-cycle. We also show in this paper that the compaction problem for a reflexive k-cycle is NP-complete for all $k \geq 4$.

Key words. computational complexity, graph, coloring, homomorphism, retraction, compaction

AMS subject classifications. 68Q25, 68R10

PII. S0097539701383522

1. Introduction.

In this paper, we study the computational complexity of a special graph coloring problem, called the compaction problem. An extended abstract of this paper appears in [Vikas, 1999]. The coloring problem is a classic problem in graph theory. The graph homomorphism problem, also called the H-coloring problem, is a generalization of the coloring problem. The compaction problem is the graph homomorphism problem with additional constraints. The open problem of Peter Winkler described above can be viewed as a special case of the compaction problem. We explain the motivation for Winkler's problem after introducing the following definitions and problems.

Definitions. The pair of vertices of an edge in a graph are called the *endpoints* of the edge. An edge with the same endpoints in a graph is called a *loop*. A vertex v of a graph is said to have a loop if vv is an edge of the graph. A *reflexive graph* is a graph in which every vertex has a loop. An *irreflexive graph* is a graph which has no loops. Any graph, in general, is a *partially reflexive graph*, meaning that its individual vertices may or may not have loops. Thus reflexive and irreflexive graphs are special partially reflexive graphs. A bipartite graph is irreflexive by definition. When we do not mention the terms reflexive, irreflexive, or bipartite, the corresponding graph may be assumed to be a partially reflexive graph. For a graph G, we use $V(G)$ and $E(G)$ to denote its vertex set and edge set, respectively. A vertex u is said to be *adjacent* to a vertex v in a graph if uv is an edge of the graph; if u is adjacent to v then v is also adjacent to u. A graph in which every two distinct vertices are adjacent is called a *complete graph*. We denote an irreflexive complete graph with k vertices by K_k. A *path* of length $k-1$ is a graph containing k distinct vertices, say $v_0, v_1, v_2, \ldots, v_{k-1}$, such that $v_0v_1, v_1v_2, \ldots, v_{k-2}v_{k-1}$ are all the nonloop edges of the graph, $k \geq 1$; we may write such a path as $v_0v_1v_2 \ldots v_{k-1}$. A *cycle* of length k, called a *k-cycle*, is a graph containing k distinct vertices, say $v_0, v_1, v_2, \ldots, v_{k-1}$, such that $v_0v_1, v_1v_2, \ldots, v_{k-2}v_{k-1}, v_{k-1}v_0$ are all the nonloop edges of the graph, $k \geq 3$;

*Received by the editors January 12, 2001; accepted for publication June 22, 2001; published electronically January 3, 2003. An extended abstract of this paper appears in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 1999.
http://www.siam.org/journals/sicomp/32-1/38352.html
†School of Computing Science, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6 (vikas@cs.sfu.ca).

we may write such a cycle as $v_0v_1v_2 \dots v_{k-1}v_0$. A *square* will be used as a synonym for a 4-cycle. For our purpose we shall denote a reflexive k-cycle by C_k.

Let G be a graph. A vertex v of G is said to be an *isolated vertex* of G if v is not adjacent to any other vertex v' of G, $v \neq v'$ (note that an isolated vertex may have a loop). A graph H is said to be a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. If H is a subgraph of G such that H contains all the edges of G whose both endpoints are in $V(H)$ then H is called the subgraph of G *induced by* $V(H)$, and we say that H is an *induced subgraph* of G. Given an induced subgraph H of G, we denote by $G - H$ the subgraph obtained by deleting from G the vertices of H together with the edges incident with them; thus $G - H$ is a subgraph of G induced by $V(G) - V(H)$. A *chordal graph* is a graph which does not contain any induced cycle of length greater than three. A *chordal bipartite graph* is a bipartite graph which does not contain any induced cycle of length greater than four. A *clique* of G is a set $K \subseteq V(G)$ such that every two distinct vertices in K are adjacent in G, i.e., the subgraph of G induced by K is a complete graph. When a set S is an argument of a mapping f, we define $f(S) = \{f(s)|s \in S\}$. The *distance* between a pair of vertices u and v in G, denoted as $d_G(u, v)$ or $d_G(v, u)$, is the length of a shortest path from u to v in G if u and v are connected in G; we define $d_G(u, v)$ (and $d_G(v, u)$) to be infinite if u and v are disconnected in G. The *diameter* of G is the maximum distance between any two vertices in G. The distance between two sets X and Y of vertices in G, denoted as $d_G(X, Y)$ or $d_G(Y, X)$, is the minimum distance between any vertex of X and any vertex of Y in G, that is, $d_G(X, Y) = \min\{d_G(x, y)|x \in X, y \in Y\}$, where $\min A$ gives the minimum element in a set A. If a set has only one vertex, we may just write the vertex instead of the set. In the following, let G and H be graphs.

A *homomorphism* $f : G \to H$, of G to H, is a mapping f of the vertices of G to the vertices of H, such that if g and g' are adjacent vertices of G then $f(g)$ and $f(g')$ are adjacent vertices of H. If there exists a homomorphism of G to H then G is said to be *homomorphic* to H. Note that for any homomorphism $f : G \to H$, if a vertex v of G has a loop then the vertex $f(v)$ of H necessarily also has a loop. Also note that if G is irreflexive then G is k-colorable if and only if G is homomorphic to K_k. Thus the concept of a homomorphism generalizes the concept of a k-colorability.

A *compaction* $c : G \to H$, of G to H, is a homomorphism of G to H, such that for every vertex x of H there exists a vertex v of G with $c(v) = x$, and for every edge hh' of H, $h \neq h'$, there exists an edge gg' of G with $c(g) = h$ and $c(g') = h'$. Notice that the first part of the definition for a compaction (the requirement for every vertex x of H) follows from the second part unless H has isolated vertices. If there exists a compaction of G to H then G is said to *compact* to H. Given a compaction $c : G \to H$, if for a vertex v of G we have $c(v) = x$, where x is a vertex of H, then we say that the vertex v of G *covers the vertex* x of H under c; and if for an edge gg' of G we have $c(\{g, g'\}) = \{h, h'\}$, where hh' is an edge of H, then we say that the edge gg' of G *covers the edge* hh' of H under c (note that in the definition of compaction, it is not necessary that a loop of H be covered by any edge of G under c).

We note that the notion of a *homomorphic image* used in [Harary, 1969] (also cf. [Hell and Miller, 1979]) coincides with the notion of a compaction in the case of irreflexive graphs (i.e., when G and H are irreflexive in the above definition for compaction).

Now suppose that H is an induced subgraph of G. A *retraction* $r : G \to H$, of G to H, is a homomorphism of G to H such that $r(h) = h$ for every vertex h of H. If there exists a retraction of G to H then G is said to *retract to* H, and H is said to be

a *retract of* G. Note that every retraction $r : G \to H$ is necessarily also a compaction.

Homomorphism, compaction, and retraction problems. The problem of deciding the existence of a homomorphism to a fixed graph H, called the *homomorphism problem for* H, also known as the H-coloring problem, and denoted as *H-COL*, asks whether or not an input graph G is homomorphic to H. If H is a graph which has a vertex h with a loop then every graph G is homomorphic to H, as we will have the homomorphism $f : G \to H$, with $f(v) = h$, for all $v \in V(G)$. Thus the problem *H-COL* is interesting only if H is irreflexive. A complete complexity classification of *H-COL* has been given in [Hell and Nesetril, 1990]. Note that the classic k-colorability problem is a special case of the problem *H-COL* when H is K_k and the input graph G is irreflexive.

The problem of deciding the existence of a compaction to a fixed graph H, called the *compaction problem for* H, and denoted as *COMP-H*, is the following:

Instance: A graph G.

Question: Does G compact to H?

Note that Winkler's problem is the problem *COMP-*C_4. We also present results for the problem *COMP-*C_k for all $k \geq 4$. The problem *COMP-*C_k can be viewed as the problem to decide whether or not it is possible to partition the vertices of a graph into k distinct nonempty sets $A_0, A_1, \ldots, A_{k-1}$, such that there is at least one edge between the pair of sets A_i and $A_{(i+1) \bmod k}$, for all $i = 0, 1, 2, \ldots, k-1$, $k > 2$, and there is no edge between any other pair of distinct sets. When both G and H are input graphs (i.e., H is not fixed), and H is reflexive, the problem of deciding whether or not G compacts to H has been studied in [Karabeg and Karabeg, 1991, 1993]. Note that unlike the problem *H-COL*, the problem *COMP-H* is still interesting if H has a loop.

The problem of deciding the existence of a retraction to a fixed graph H, called the *retraction problem for* H, and denoted as *RET-H*, asks whether or not an input graph G, containing H as an induced subgraph, retracts to H. Retraction problems have been of continuing interest in graph theory for a long time and have been studied in various literature including [Hell, 1972], [Hell, 1974], [Nowakowski and Rival, 1979], [Pesch and Poguntke, 1985], [Bandelt, Dahlmann, and Schutte, 1987], [Hell and Rival, 1987], [Pesch, 1988], [Feder and Winkler, 1988], [Bandelt, Farber, and Hell, 1993], [Feder and Hell, 1998], [Feder, Hell, and Huang, 1999].

Note that the graph H for the problems *H-COL*, *COMP-H*, and *RET-H* is assumed to be fixed by default even if not explicitly mentioned.

Motivation. It has been proved in [Feder and Winkler, 1988] that *RET-*C_4 is NP-complete. This has also been proved independently by G. MacGillivray in 1988 (personal communication, cf. [Feder and Hell, 1998]). The transformation used in [Feder and Winkler, 1988] is from the satisfiability problem; this was proved there in order to show that *RET-H* is not polynomial time solvable for all graphs H unless P = NP. The transformation used by MacGillivray is from the 4-colorability problem.

It is not difficult to show that for every fixed graph H, if *RET-H* is solvable in polynomial time then *COMP-H* is also solvable in polynomial time. Is the converse true? This was also asked by Winkler in the context of reflexive graphs (personal communication, cf. [Feder and Winkler, 1988]), and this was the general problem that motivated Winkler. As mentioned above, *RET-*C_4 is NP-complete. It turns out that for any reflexive graph H, other than C_4, with at most four vertices, *RET-H* is polynomial time solvable. In other words, as mentioned in [Feder and Winkler, 1988], the unique smallest reflexive graph H for which *RET-H* is NP-complete is

C_4. Therefore, with respect to the preceding question, Winkler asked specifically the following question in 1988 (personal communication, cf. [Feder and Winkler, 1988]) which has been a popular open problem: Is *COMP-C_4* NP-complete? We show in this paper that *COMP-C_4* is NP-complete. To show this, we give a transformation from *RET-C_4* to *COMP-C_4*, using our technique explained below.

Our generic technique. We describe here a technique that we developed to use in our proofs. Let a graph G containing H as an induced subgraph be an instance of *RET-H*. When we give a transformation from *RET-H* to *COMP-H*, we apply the following technique. We construct in time polynomial in the size of G, a graph G' (containing G as an induced subgraph) such that the following statements (i), (ii), and (iii) are equivalent:

(i) G retracts to H.

(ii) G' retracts to H.

(iii) G' compacts to H.

Thus if *RET-H* is NP-complete, this shows that *COMP-H* is also NP-complete. It is this technique that we have used throughout, when giving a transformation from *RET-H* to *COMP-H*, for any graph H. Thus our technique turned out to be a generic one. We prove the equivalence of the above statements by showing that (i) is equivalent to (ii), and (ii) is equivalent to (iii).

Results. We now first mention the results known for *RET-C_k*. Feder extended the proof of NP-completeness of *RET-C_4* showing that *RET-C_k* is NP-complete for all $k \geq 4$ (personal communication through Hell, cf. [Feder and Hell, 1998]). This was also proved independently by G. MacGillivray in 1988 (personal communication). The transformation used by both Feder and MacGillivray is from the k-colorability problem. For the case of a reflexive 3-cycle, it is easily seen that *RET-C_3*, and hence *COMP-C_3*, are both polynomial time solvable. In fact, when H is a reflexive chordal graph (which includes C_3), the problem *RET-H* is shown to be polynomial time solvable in [Feder and Hell, 1998], and hence *COMP-H* is also polynomial time solvable.

In this paper, we show that *COMP-C_k* is NP-complete for all $k \geq 4$. We do this by giving a transformation from *RET-C_k* to *COMP-C_k*, using the technique described above, for all $k \geq 4$. For proving NP-completeness of *COMP-C_k*, for all $k \geq 4$, we provide four categories of construction depending on whether $k = 4m, 4m+1, 4m+2$, or $4m+3$, for some integer $m \geq 1$.

Since every retraction is also a compaction, the equivalence of (ii) and (iii) in the above technique implies that if G' compacts to H then there exists a compaction $c : G' \to H$ which is also a retraction. Hence, in our definition of compaction, if we require a compaction to also cover loops then the equivalence of (i), (ii), and (iii) in the above technique implies that if *RET-H* is NP-complete then *COMP-H* remains NP-complete under this varied definition of compaction also. Thus, with the above technique, our NP-completeness results for *COMP-H* in this paper also hold under this varied definition of compaction.

In the figures in this paper, we shall not be depicting any edge vh of G, with $v \in V(G - H)$ and $h \in V(H)$, where G is any graph containing H as an induced subgraph, i.e., G is an instance of *RET-H*. Also, in the figures in this paper, we shall not be depicting any loops. For the problem *COMP-H*, in the theorems in this paper, it is sufficient to remember that there is a loop on each vertex of the graph H. The presence or absence of a loop on any other vertex of any graph is immaterial for these theorems.

Although the NP-completeness of deciding the existence of a compaction to a reflexive square is a special case of the NP-completeness of deciding the existence of a compaction to a reflexive k-cycle, for all $k \geq 4$, we first present, in section 2, the proof for the case of a reflexive square which is easier to follow and will be helpful in understanding the proof for the case of a reflexive k-cycle, for all $k \geq 4$, which we present next in section 3. In section 4, we present our results for compaction to irreflexive cycles without including the proofs.

2. Compaction to a reflexive square. We prove the following theorem in this section.

THEOREM 2.1. *The problem of deciding the existence of a compaction to a reflexive square is NP-complete.*

Proof. Let H be the reflexive square in Figure 2.1.

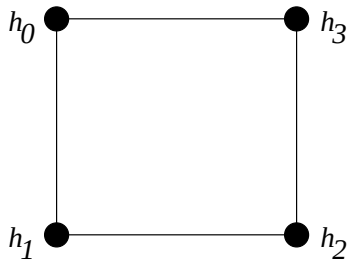


FIG. 2.1. H.

We shall prove that the problem of deciding the existence of a compaction to H, i.e., the problem *COMP-H*, is NP-complete. Clearly, *COMP-H* is in NP. We give a polynomial transformation from *RET-H* to *COMP-H*. As mentioned earlier, the problem *RET-H* is NP-complete. Let G be a graph with H as an induced subgraph of G, i.e., let G be an instance of *RET-H*. We construct in time polynomial in the size of G, a graph G' (containing G as an induced subgraph) such that the statements (i), (ii), and (iii) mentioned in our technique in section 1 are equivalent. Since *RET-H* is NP-complete, this shows that *COMP-H* is also NP-complete. We prove that (i) is equivalent to (ii), and (ii) is equivalent to (iii), in Lemma 2.1.1 and Lemma 2.1.2, respectively.

The construction of G' is as follows. For each vertex v in $V(G-H)$, we add to G three distinct new vertices: u_v adjacent to v, h_0, h_1; w_v adjacent to v, h_2, h_3, u_v; and y_v adjacent to h_0, h_2, u_v, w_v. See Figure 2.2. Note that there could be edges in G from v to some vertices of H but as mentioned earlier, in Figure 2.2 and all subsequent figures in this paper, we are not depicting these edges. Furthermore, we add the edges $u_v u_{v'}$ and $w_v w_{v'}$ for all $v, v' \in V(G-H), v \neq v'$.

Before completing the construction of G', we choose one fixed direction for every edge of $G-H$. By convention, this chosen direction will be implied by the order of the endpoints, when an edge is specified in this proof, i.e., when we mention an edge vv' of $G-H$ in this proof, we assume that the chosen direction is from v to v'. For every edge vv' in $E(G-H)$, $v \neq v'$, we add a new vertex $x_{vv'}$ adjacent to v, v', u_v, and $w_{v'}$. (Note how the choice of a direction for the edge vv' affects the adjacencies from $x_{vv'}$). See Figure 2.3.

This completes the construction of G'. For an edge $vv' \in E(G-H)$, the construction of G' is illustrated in Figure 2.4. We now prove the following two lemmas in order to prove the theorem.

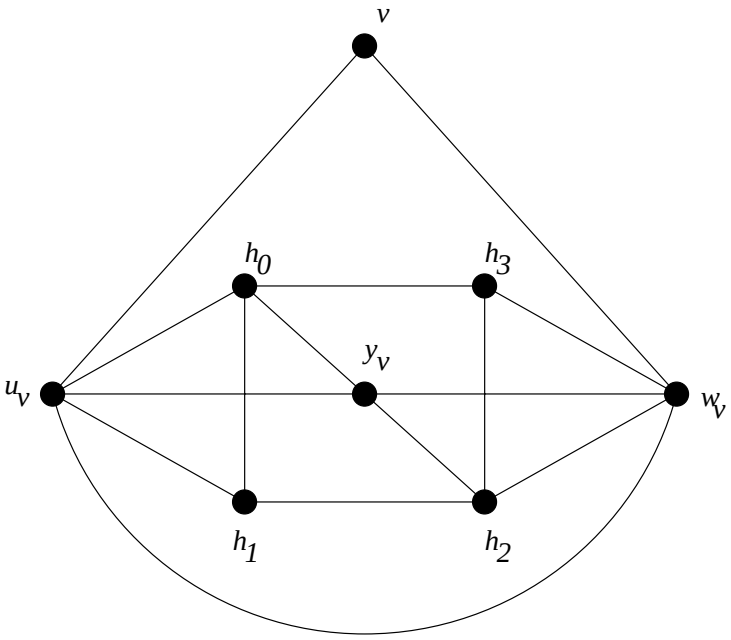


FIG. 2.2. *Construction of G' for a vertex v in $G - H$.*

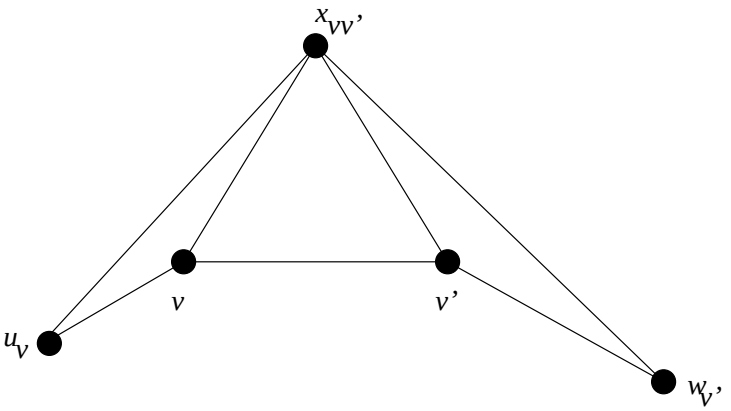


FIG. 2.3. *Construction of G' for an edge vv' in $G - H$.*

LEMMA 2.1.1. *G retracts to H if and only if G' retracts to H.*

Proof. If G' retracts to H then it is clear that G retracts to H since G is a subgraph of G'. Now suppose that $r : G \to H$ is a retraction. We define a retraction $r' : G' \to H$ as follows.

For each vertex v of the graph G, we define

$r'(v) = r(v)$.

For the vertices u_v, w_v, and y_v of G', with $v \in V(G - H)$, we define

$r'(u_v) = h_1$, $r'(w_v) = h_2$, and $r'(y_v) = h_1$, if $r(v) = h_1$ or h_2, and

$r'(u_v) = h_0$, $r'(w_v) = h_3$, and $r'(y_v) = h_3$, if $r(v) = h_0$ or h_3.

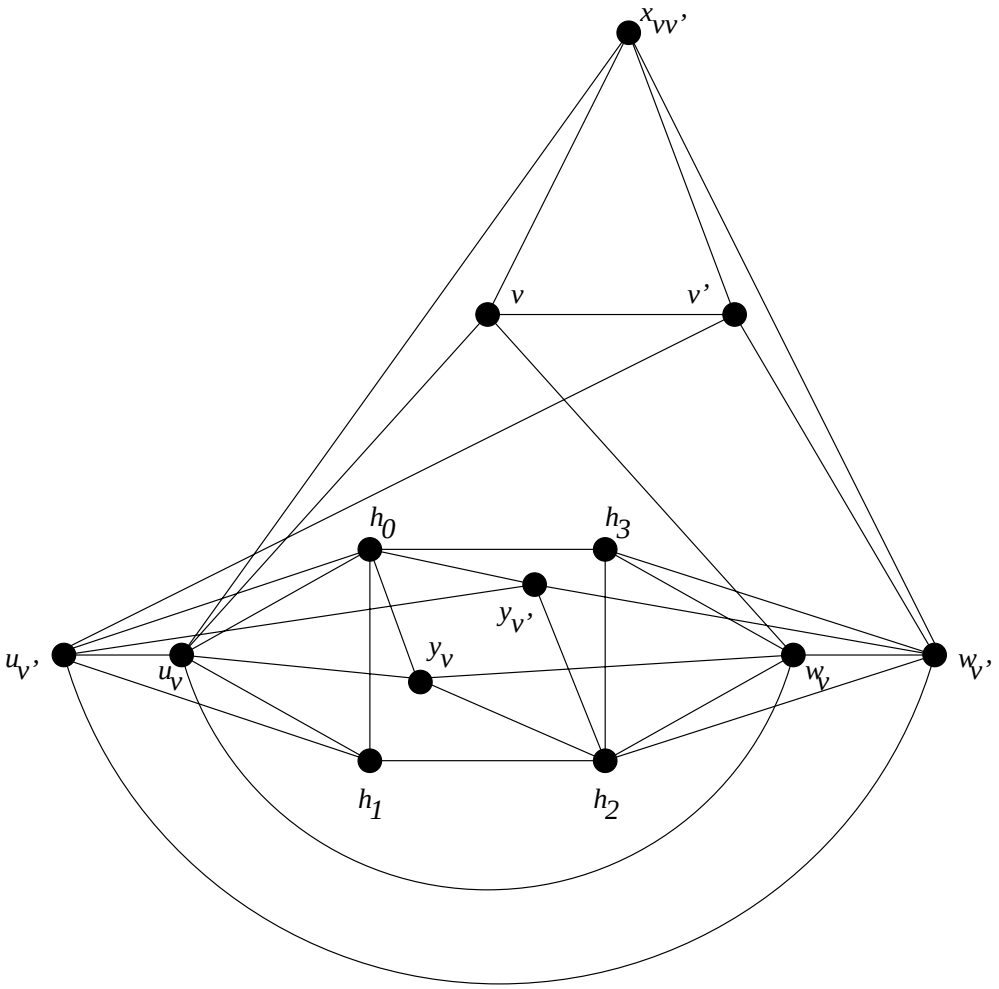


FIG. 2.4. *Construction of G', with $vv' \in E(G-H)$.*

For the vertex $x_{vv'}$ of G', with $vv' \in E(G-H)$, we define

$r'(x_{vv'}) = r(v)$, if $r(v) = h_2$ or h_3, and

$r'(x_{vv'}) = r(v')$, if $r(v) = h_0$ or h_1.

We now verify that $r' : G' \to H$ is indeed a homomorphism (and hence a retraction). We shall do this by considering all the edges ab of G' and proving that $r'(a)r'(b)$ is an edge of H.

Consider first an edge vv', with $vv' \in E(G)$. We have $r'(v)r'(v') = r(v)r(v')$. Hence $r'(v)r'(v')$ is an edge of H (as $r : G \to H$ is a homomorphism).

Next consider an edge $u_v u_{v'}$, with $v, v' \in V(G-H)$. We have that $r'(u_v)$ and $r'(u_{v'})$ are h_0 or h_1 depending on the values of $r(v)$ and $r(v')$, respectively. Hence $r'(u_v)r'(u_{v'})$ is always an edge of H. For the edges $u_v h_0$ and $u_v h_1$, with $v \in V(G-H)$, we argue similarly, since $r'(h_0) = r(h_0) = h_0$ and $r'(h_1) = r(h_1) = h_1$. In a similar way, we also argue for the edges $w_v w_{v'}, w_v h_2$, and $w_v h_3$, with $v, v' \in V(G-H)$.

Now consider an edge $u_v v$, with $v \in V(G-H)$. We have $r'(v) = r(v)$, and if $r(v) = h_1$ or h_2 then $r(u_v) = h_1$, otherwise $r(u_v) = h_0$. Thus $r'(u_v)r'(v)$ is always an

edge of H. Similarly, we argue for an edge $w_v v$, with $v \in V(G-H)$.

Take now an edge $y_v h_0$, with $v \in V(G-H)$. We have $r'(h_0) = r(h_0) = h_0$, and depending on the value of $r(v)$, $r'(y_v) = h_1$ or h_3. Thus $r'(y_v)r'(h_0)$ is always an edge of H. Similarly, we argue for an edge $y_v h_2$, with $v \in V(G-H)$.

Now consider an edge $u_v w_v$, with $v \in V(G-H)$. If $r(v) = h_1$ or h_2, we have $r'(u_v) = h_1$ and $r'(w_v) = h_2$. If $r(v) = h_0$ or h_3, we have $r'(u_v) = h_0$ and $r'(w_v) = h_3$. Thus $r'(u_v)r'(w_v)$ is always an edge of H.

Now take an edge $y_v u_v$, with $v \in V(G-H)$. If $r(v) = h_1$ or h_2, we have $r'(u_v) = h_1$ and $r'(y_v) = h_1$. If $r(v) = h_0$ or h_3, we have $r'(u_v) = h_0$ and $r'(y_v) = h_3$. Thus $r'(y_v)r'(u_v)$ is always an edge of H. Similarly, we argue for an edge $y_v w_v$, with $v \in V(G-H)$.

Consider now an edge $x_{vv'}v$, with $vv' \in E(G-H)$. We have $r'(x_{vv'}) = r(v)$ or $r(v')$, and $r'(v) = r(v)$. Since $r(v)r(v')$ must be an edge of H, $r'(x_{vv'})r'(v)$ is always an edge of H. Similarly, we argue for an edge $x_{vv'}v'$, with $vv' \in E(G-H)$.

Next consider an edge $x_{vv'}u_v$, with $vv' \in E(G-H)$. First suppose that $r(v) = h_2$ or h_3. Then $r'(x_{vv'}) = r(v)$. We have already proved that $r'(v)r'(u_v) = r(v)r'(u_v)$ is an edge of H. Hence $r'(x_{vv'})r'(u_v) = r(v)r'(u_v)$ is an edge of H. Now suppose that $r(v) = h_0$ or h_1. Then $r'(x_{vv'}) = r(v')$. If $r(v) = h_0$ then $r'(u_v) = h_0 = r(v)$ and $r'(x_{vv'})r'(u_v) = r(v')r(v)$ is an edge of H. If $r(v) = h_1$ then $r'(u_v) = h_1 = r(v)$ and $r'(x_{vv'})r'(u_v) = r(v')r(v)$ is again an edge of H. Thus we have proved that $r'(x_{vv'})r'(u_v)$ is always an edge of H.

Finally, consider an edge $x_{vv'}w_{v'}$, with $vv' \in E(G-H)$. First suppose that $r(v) = h_2$ or h_3. Then $r'(x_{vv'}) = r(v)$. We have $r'(w_{v'}) = h_2$ or h_3 depending on the value of $r(v')$. Thus $r(v)r'(w_{v'})$ is an edge of H, and hence $r'(x_{vv'})r'(w_{v'}) = r(v)r'(w_{v'})$ is an edge of H. Now suppose that $r(v) = h_0$ or h_1. Then $r'(x_{vv'}) = r(v')$. We have already proved that $r'(v')r'(w_{v'}) = r(v')r'(w_{v'})$ is an edge of H. Hence $r'(x_{vv'})r'(w_{v'}) = r(v')r'(w_{v'})$ is an edge of H. Thus we have proved that $r'(x_{vv'})r'(w_{v'})$ is always an edge of H.

Thus we have proved that $r' : G' \to H$ is a homomorphism. Since $r'(h) = r(h) = h$, for all $h \in V(H)$, $r' : G' \to H$ is a retraction, and the lemma is proved. □

LEMMA 2.1.2. *G' retracts to H if and only if G' compacts to H.*

Proof. If G' retracts to H then by definition G' compacts to H. Now suppose that $c : G' \to H$ is a compaction. We let $U = \{u_v | v \in V(G-H)\} \cup \{h_0, h_1\}$ and $W = \{w_v | v \in V(G-H)\} \cup \{h_2, h_3\}$. Note that U and W are cliques in G'.

Since U is a clique in G', $c(U)$ must be a clique in H. Thus $c(U)$ has either one or two vertices. Similarly, $c(W)$ has either one or two vertices. We shall prove that $c(U)$ and $c(W)$ both have two vertices.

Suppose that $c(U)$ has only one vertex. Without loss of generality, let $c(U) = \{h_0\}$. We can assume this, as due to symmetry of vertices in H, we can always redefine the compaction c so that $c(U) = \{h_0\}$. As every vertex of G' is adjacent to a vertex in U, no vertex a of G' can have $c(a) = h_2$. Thus $c(U)$ must have two vertices. Similarly, $c(W)$ must also have two vertices.

Thus both $c(U)$ and $c(W)$ are cliques of size two in H. Without loss of generality, suppose that $c(U) = \{h_0, h_1\}$ (due to symmetry). We first prove that $c(W) = \{h_2, h_3\}$. Let some edge ab of G' cover the edge h_2h_3 of H under c (indeed there exists such an edge in G', as $c : G' \to H$ is a compaction). Then both a and b are not in U, as $c(U) = \{h_0, h_1\}$. Thus a and b are among the vertices h_2, h_3; w_v, y_v, v, with $v \in V(G-H)$; and $x_{vv'}$, with $vv' \in E(G-H)$. Suppose that $c(W) \neq \{h_2, h_3\}$. Then no edge with both endpoints in W covers the edge h_2h_3 of H under c, and hence not

both a and b belong to W. Thus ab must be an edge among vw_v, y_vw_v, y_vh_2, with $v \in V(G-H)$; $vv', vx_{vv'}, v'x_{vv'}$, and $w_{v'}x_{vv'}$, with $vv' \in E(G-H)$. Further, if vh_2 or vh_3 is an edge of G, for some vertex $v \in V(G-H)$, then we need to include such an edge also for ab. We shall consider each of these possible edges for ab and show that they do not cover the edge h_2h_3 under c, i.e., $c(\{a,b\}) \neq \{h_2,h_3\}$, implying that $c(W) = \{h_2,h_3\}$.

Consider first an edge vw_v, with $v \in V(G-H)$. If $c(\{v,w_v\}) = \{h_2,h_3\}$ then it must be that $c(u_v) = h_2$ or h_3, as u_v is adjacent to v and w_v. This is a contradiction, as $c(u_v) = h_0$ or h_1 by assumption. A similar argument applies to an edge $vx_{vv'}$, with $vv' \in E(G-H)$; and an edge y_vw_v, with $v \in V(G-H)$.

Next consider an edge y_vh_2, with $v \in V(G-H)$. If $c(\{y_v,h_2\}) = \{h_2,h_3\}$ then $c(w_v) = h_2$ or h_3, as w_v is adjacent to y_v and h_2. Since $c(W) \neq \{h_2,h_3\}$, it must be that $c(w_v) = c(h_2)$. This implies that $c(\{y_v,w_v\}) = \{h_2,h_3\}$, which as noted above does not hold. Similarly, for the possible edges vh_2 and vh_3, with $v \in V(G-H)$, if $c(v,h_2) = \{h_2,h_3\}$ or $c(v,h_3) = \{h_2,h_3\}$ then we will have $c(v,w_v) = \{h_2,h_3\}$, which we have shown above is not possible.

Now consider an edge $vv' \in E(G-H)$. Suppose that $c(\{v,v'\}) = \{h_2,h_3\}$. Without loss of generality, let $c(v) = h_2$ and $c(v') = h_3$ (due to symmetry). Since u_v and $u_{v'}$ are adjacent to v and v', respectively, and $c(U) = \{h_0,h_1\}$, we have $c(u_v) = h_1$ and $c(u_{v'}) = h_0$. Since $c(w_v)$ must be adjacent to $c(v) = h_2$ and $c(u_v) = h_1$, we have $c(w_v) \in \{h_1,h_2\}$. Also, $c(w_{v'})$ must be adjacent to $c(u_{v'}) = h_0$ and $c(v') = h_3$. Hence $c(w_{v'}) \in \{h_0,h_3\}$. Since w_v is adjacent to $w_{v'}$ in G', and $c(W) \neq \{h_2,h_3\}$, this implies that $c(w_v) = h_1$ and $c(w_{v'}) = h_0$. Now $c(x_{vv'})$ must be adjacent to $c(w_{v'}) = h_0$, $c(u_v) = h_1$, $c(v) = h_2$, and $c(v') = h_3$, which is impossible.

Consider now an edge $v'x_{vv'}$, with $vv' \in E(G-H)$. If $c(\{v',x_{vv'}\}) = \{h_2,h_3\}$ then $c(v) = h_2$ or h_3, as v is adjacent to v' and $x_{vv'}$. This implies that either $c(\{v,v'\}) = \{h_2,h_3\}$ or $c(\{v,x_{vv'}\}) = \{h_2,h_3\}$, both of which we have already proved do not hold.

Finally, consider an edge $w_{v'}x_{vv'}$, with $vv' \in E(G-H)$. If $c(\{w_{v'},x_{vv'}\}) = \{h_2,h_3\}$ then $c(v') = h_2$ or h_3, as v' is adjacent to $w_{v'}$ and $x_{vv'}$. This implies that either $c(\{v',w_{v'}\}) = \{h_2,h_3\}$ or $c(\{v',x_{vv'}\}) = \{h_2,h_3\}$, both of which we have already proved do not hold.

This completes the proof that $c(W) = \{h_2,h_3\}$. We now prove that $c(h_0) \neq c(h_1)$. Suppose to the contrary that $c(h_0) = c(h_1)$. We know that $c(h_0), c(h_1) \in \{h_0,h_1\}$ (as $c(U) = \{h_0,h_1\}$). Without loss of generality, let $c(h_0) = c(h_1) = h_0$ (due to symmetry). Since $c(U) = \{h_0,h_1\}$, $c(u_v) = h_1$ for some vertex v of $G-H$. Since w_v and h_2 are adjacent to u_v and h_1, respectively, and $c(W) = \{h_2,h_3\}$, we have $c(w_v) = h_2$ and $c(h_2) = h_3$. Now $c(y_v)$ must be adjacent to $c(h_0) = h_0$, $c(u_v) = h_1$, $c(w_v) = h_2$, and $c(h_2) = h_3$, which is impossible.

Thus $c(h_0) \neq c(h_1)$, i.e., $c(\{h_0,h_1\}) = \{h_0,h_1\}$. Without loss of generality, suppose that $c(h_0) = h_0$ and $c(h_1) = h_1$ (due to symmetry). Since h_2 and h_3 are adjacent to h_1 and h_0, respectively, and $c(W) = \{h_2,h_3\}$, we have $c(h_2) = h_2$ and $c(h_3) = h_3$. Thus $c : G' \to H$ is a retraction, and the lemma is proved. □

We have thus proved Theorem 2.1. □

3. Compaction to a reflexive k-cycle. We prove the following theorem in this section.

THEOREM 3.1. *The problem of deciding the existence of a compaction to a reflexive k-cycle is NP-complete for all $k \geq 4$.*

Proof. Let H be the reflexive k-cycle $h_0h_1 \ldots h_{k-1}h_0$, with $k \geq 4$. See Figure 3.1.

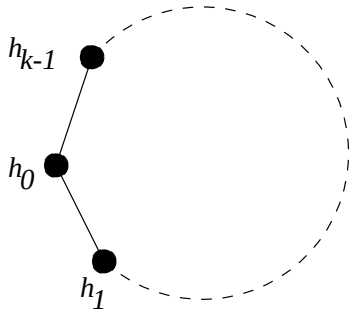


FIG. 3.1. H.

We shall prove that the problem of deciding the existence of a compaction to H, i.e., the problem *COMP-H*, is NP-complete. Clearly, the problem *COMP-H* is in NP. We give a polynomial transformation from the problem *RET-H* to *COMP-H*. As mentioned earlier, the problem *RET-H* is NP-complete. Let a graph G containing H as an induced subgraph be an instance of *RET-H*. We construct in time polynomial in the size of G, a graph G' (containing G as an induced subgraph) such that the statements (i), (ii), and (iii) mentioned in our technique in section 1 are equivalent. Since *RET-H* is NP-complete, this shows that *COMP-H* is also NP-complete. In analogy to the proof of Theorem 2.1 for a reflexive square, we prove that (i) is equivalent to (ii), and (ii) is equivalent to (iii), in two separate lemmas.

Assume that $k = 4m, 4m+1, 4m+2$, or $4m+3$, for some integer $m \geq 1$. Thus for any k we have $m = \lfloor k/4 \rfloor$. We also let $n = \lfloor k/2 \rfloor$. It may be helpful to note that $n - m = m + 1$ for $k = 4m + 2$ and $k = 4m + 3$, and $n - m = m$ for $k = 4m$ and $k = 4m + 1$. There are four categories of construction for G', depending on whether $k = 4m, 4m+1, 4m+2$, or $4m+3$. We include here a detailed proof of the theorem for $k = 4m + 2$ and $k = 4m + 3$. The proof for $k = 4m$ and $k = 4m + 1$ is similar to the proof for $k = 4m + 2$ and $k = 4m + 3$, respectively. Hence we will make remarks and refer to the cases $k = 4m + 2$ and $k = 4m + 3$ when proving the theorem for the cases $k = 4m$ and $k = 4m + 1$, respectively. We shall assume that all arithmetic operations are done modulo k unless otherwise convenient.

We prove the theorem in the cases $k = 4m + 2$, $k = 4m$, $k = 4m + 3$, and $k = 4m + 1$ in separate sections.

Proof of Theorem 3.1 in the case $k = 4m + 2$. The construction of G' is as follows. For each vertex v in $V(G - H)$, we add to G three vertex disjoint paths U_v, W_v, and Y_v, each containing $n - 1$ new vertices. Let $U_v = u_1^v u_2^v \dots u_{n-1}^v$, $W_v = w_1^v w_2^v \dots w_{n-1}^v$, and $Y_v = y_1^v y_2^v \dots y_{n-1}^v$, with $v \in V(G - H)$. We add the edges $u_1^v h_0$, $u_1^v h_1$, $v u_{n-1}^v$, $w_1^v h_n$, $w_1^v h_{n+1}$, $v w_{n-1}^v$, $u_{m+1}^v w_{m+1}^v$, $y_1^v h_0$, $y_{n-1}^v h_n$, $u_1^v y_1^v$, and $w_1^v y_{n-1}^v$, with $v \in V(G - H)$ (note that h_n is the vertex opposite to h_0 in H). For each $v \in V(G - H)$, we also add a new vertex a_v adjacent to u_m^v and w_m^v. For convenience, in the proof of the first lemma, we also use h_k to denote the vertex h_0. See Figure 3.2. Furthermore, we add the edges $u_1^v u_1^{v'}$ and $w_1^v w_1^{v'}$ for all $v, v' \in V(G - H)$, $v \neq v'$.

As for a reflexive square in Theorem 2.1, we choose one fixed direction for every edge of $G - H$, and when we mention an edge vv' of $G - H$ in this proof, we assume that the chosen direction is from v to v'. For every edge $vv' \in E(G - H)$, $v \neq v'$, we add a new vertex $x_{vv'}$ adjacent to v, v', u_{n-1}^v, and $w_{n-1}^{v'}$. See Figure 3.3.

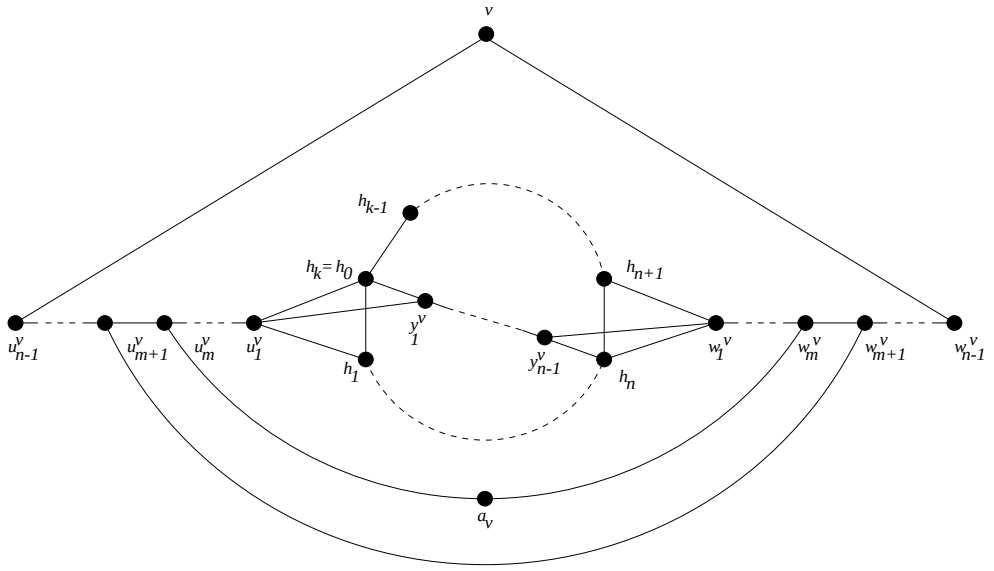


FIG. 3.2. *Construction of G' for a vertex v in $G - H$, with $k = 4m + 2$.*

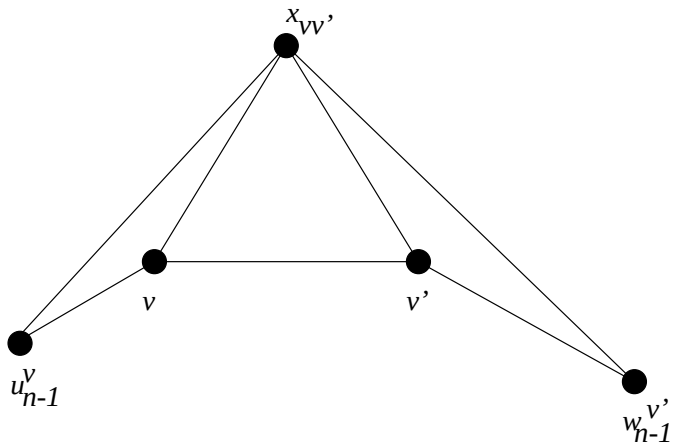


FIG. 3.3. *Construction of G' for an edge vv' in $G - H$.*

This completes the construction of G'. We now prove the following two lemmas in order to prove the theorem for $k = 4m + 2$.

LEMMA 3.1.1. *G retracts to H if and only if G' retracts to H.*

Proof. If G' retracts to H then clearly G retracts to H, as G is a subgraph of G'. Now suppose that $r : G \to H$ is a retraction. Below, we define a retraction $r' : G' \to H$. As we go along the definition of r', we shall be considering the edges ab of G', showing that $r'(a)r'(b)$ is indeed an edge of H (as required for $r' : G' \to H$ to be a homomorphism). Recall that the edges of G' are ab, $u_i^v u_{i+1}^v$, $w_i^v w_{i+1}^v$, $y_i^v y_{i+1}^v$, $u_1^v h_0$, $u_1^v h_1$, $w_1^v h_n$, $w_1^v h_{n+1}$, $v u_{n-1}^v$, $v w_{n-1}^v$, $u_{m+1}^v w_{m+1}^v$, $a_v u_m^v$, $a_v w_m^v$, $y_1^v h_0$, $y_{n-1}^v h_n$, $u_1^v y_1^v$, $w_1^v y_{n-1}^v$, $u_1^v u_1^{v'}$, $w_1^v w_1^{v'}$, $g x_{gg'}$, $g' x_{gg'}$, $x_{gg'} u_{n-1}^g$, and $x_{gg'} w_{n-1}^{g'}$, with $ab \in E(G)$, $v, v' \in V(G - H)$, $gg' \in E(G - H)$, $i = 1, 2, \ldots, n - 2$.

For each vertex v of the graph G, we define

$r'(v) = r(v)$.

Thus for an edge vv' of G', with $vv' \in E(G)$, we have that $r'(v)r'(v') = r(v)r(v')$ is an edge of H.

We now fix a vertex $v \in V(G - H)$ for defining r' for the vertices of U_v, W_v, and Y_v, and for the vertex a_v. Let $r(v) = h_j$. For the purpose of this lemma, as mentioned earlier, we use h_0 and h_k to denote the same vertex, and use the subscript 0 or k as convenient. We shall define r' for the said vertices when $1 \leq j \leq n$ and when $n + 1 \leq j \leq k$.

First assume that $1 \leq j \leq n$.

For the vertices of U_v, we define r' as follows.

If $j \leq m$ then we define

$r'(u_i^v) = h_i$ for all $i = 1, 2, \ldots, m$,

$r'(u_i^v) = h_{n-i}$ for all $i = m + 1, m + 2, \ldots, n - j$,

$r'(u_i^v) = h_j$ for all $i = n - j + 1, n - j + 2, \ldots, n - 1$.

Thus for the edges $u_1^v h_0$, $u_1^v h_1$, and vu_{n-1}^v of G', $r'(u_1^v)r'(h_0) = h_1h_0$, $r'(u_1^v)r'(h_1) = h_1h_1$, and $r'(v)r'(u_{n-1}^v) = h_jh_j$ are, respectively, the edges of H, when $j \leq m$. Clearly, for the edge $u_i^v u_{i+1}^v$ of G', $r'(u_i^v)r'(u_{i+1}^v)$ is an edge of H for all $i = 1, 2, \ldots, n - 2$.

If $j > m$ then we define

$r'(u_i^v) = h_i$ for all $i = 1, 2, \ldots, j - 1$,

$r'(u_i^v) = h_j$ for all $i = j, j + 1, \ldots, n - 1$.

Thus with $j > m$, we have $r'(u_{n-1}^v) = h_{j-1}$ if $j = n$, and $r'(u_{n-1}^v) = h_j$ if $j < n$. Hence for an edge vu_{n-1}^v of G', $r'(v)r'(u_{n-1}^v) = h_jr'(u_{n-1}^v)$ is an edge of H, when $j > m$. Also, for the edges ab of G' among $u_1^v h_0$, $u_1^v h_1$, and $u_i^v u_{i+1}^v$, for all $i = 1, 2, \ldots, n - 2$, clearly $r'(a)r'(b)$ is an edge of H, when $j > m$.

For the vertices of W_v, we define r' as follows.

If $j \geq n - m + 1$ then we define

$r'(w_i^v) = h_{n-i+1}$ for all $i = 1, 2, \ldots, m$,

$r'(w_i^v) = h_{i+1}$ for all $i = m + 1, m + 2, \ldots, j - 1$,

$r'(w_i^v) = h_j$ for all $i = j, j + 1, \ldots, n - 1$.

If $j < n - m + 1$ then we define

$r'(w_i^v) = h_{n-i+1}$ for all $i = 1, 2, \ldots, n - j$,

$r'(w_i^v) = h_j$ for all $i = n - j + 1, n - j + 2, \ldots, n - 1$.

As for the case of U_v, it can be verified that for the edges ab of G' among $w_1^v h_n$, $w_1^v h_{n+1}$, vw_{n-1}^v, and $w_i^v w_{i+1}^v$, for all $i = 1, 2, \ldots, n - 2$, $r'(a)r'(b)$ is always an edge of H.

Now we show that for an edge $u_{m+1}^v w_{m+1}^v$ of G', $r'(u_{m+1}^v)r'(w_{m+1}^v)$ is an edge of H. We note that $r'(w_{m+1}^v) = h_{m+2}$ if $j \geq n-m+1$, and $r'(w_{m+1}^v) = h_{n-m} = h_{m+1}$ if $j < n-m+1$. Also, we have $r'(u_{m+1}^v) = h_{n-m-1} = h_m$ if $j \leq m$, and $r'(u_{m+1}^v) = h_{m+1}$ if $j > m$. Thus, if $j < n - m + 1 = m + 2$ then $r'(u_{m+1}^v)r'(w_{m+1}^v) = r'(u_{m+1}^v)h_{m+1}$ is an edge of H (as $r'(u_{m+1}^v) = h_m$ or h_{m+1}). If $j \geq n - m + 1 = m + 2$ then $r'(u_{m+1}^v)r'(w_{m+1}^v) = h_{m+1}h_{m+2}$ is an edge of H. Thus under all possibilities, we have shown that $r'(u_{m+1}^v)r'(w_{m+1}^v)$ is an edge of H.

For the vertex a_v, we define

$r'(a_v) = h_{m+1}$.

Thus for the edges $a_v u_m^v$ and $a_v w_m^v$ of G', $r'(a_v)r'(u_m^v) = h_{m+1}h_m$ and $r'(a_v)r'(w_m^v) = h_{m+1}h_{n-m+1} = h_{m+1}h_{m+2}$ are, respectively, the edges of H.

For the vertices of Y_v, we define

$r'(y_i^v) = h_i$ for all $i = 1, 2, \ldots, n - 1$.

Thus for the edges $u_1^v y_1^v$ and $w_1^v y_{n-1}^v$ of G', $r'(u_1^v)r'(y_1^v) = h_1h_1$ and $r'(w_1^v)r'(y_{n-1}^v) = h_nh_{n-1}$ are, respectively, the edges of H. Also, for the edges ab of G' among $y_1^v h_0$, $y_{n-1}^v h_n$, and $y_i^v y_{i+1}^v$, for all $i = 1, 2, \ldots, n-2$, clearly $r'(a)r'(b)$ is an edge of H.

Now assume that $n+1 \leq j \leq k$.

Similar to the previous case, where $1 \leq j \leq n$, it can be verified that for the edges ab of G' considered in the previous case, $r'(a)r'(b)$ is indeed an edge of H in this case also, where $n+1 \leq j \leq k$.

For the vertices of U_v, we define r' as follows.

If $j \geq k-m+1$ then we define

$r'(u_i^v) = h_{k-i+1}$ for all $i = 1, 2, \ldots, m$,

$r'(u_i^v) = h_{n+i+1}$ for all $i = m+1, m+2, \ldots, j-n-1$,

$r'(u_i^v) = h_j$ for all $i = j-n, j-n+1, \ldots, n-1$.

If $j < k-m+1$ then we define

$r'(u_i^v) = h_{k-i+1}$ for all $i = 1, 2, \ldots, k-j$,

$r'(u_i^v) = h_j$ for all $i = k-j+1, k-j+2, \ldots, n-1$.

For the vertices of W_v, we define r' as follows.

If $j \leq n+m$ then we define

$r'(w_i^v) = h_{n+i}$ for all $i = 1, 2, \ldots, m$,

$r'(w_i^v) = h_{k-i}$ for all $i = m+1, m+2, \ldots, k-j$,

$r'(w_i^v) = h_j$ for all $i = k-j+1, k-j+2, \ldots, n-1$.

If $j > n+m$ then we define

$r'(w_i^v) = h_{n+i}$ for all $i = 1, 2, \ldots, j-n-1$,

$r'(w_i^v) = h_j$ for all $i = j-n, j-n+1, \ldots, n-1$.

For the vertex a_v, we define

$r'(a_v) = h_{k-m}$.

For the vertices of Y_v, we define

$r'(y_i^v) = h_{k-i}$ for all $i = 1, 2, \ldots, n-1$.

This completes the definition of r' for the vertices of U_v, W_v, and Y_v and for the vertex a_v.

Consider now the edges $u_1^v u_1^{v'}$ and $w_1^v w_1^{v'}$ of G', with $v, v' \in V(G-H)$. From the definition of r', we have that $r'(u_1^v)$ and $r'(u_1^{v'})$ are h_1 or h_0, and $r'(w_1^v)$ and $r'(w_1^{v'})$ are h_n or h_{n+1}. Thus $r'(u_1^v)r'(u_1^{v'})$ and $r'(w_1^v)r'(w_1^{v'})$ are always the edges of H.

For the vertex $x_{vv'}$, with $vv' \in E(G-H)$, we define

$r'(x_{vv'}) = r(v)$, if $r(v) \notin \{h_0, h_1\}$, and

$r'(x_{vv'}) = r(v')$, if $r(v) \in \{h_0, h_1\}$.

We now consider the edges of G' associated with $x_{vv'}$, with $vv' \in E(G-H)$. For the edges $vx_{vv'}$ and $v'x_{vv'}$ of G', we argue as for the case of a reflexive square in Lemma 2.1.1 that $r'(v)r'(x_{vv'})$ and $r'(v')r'(x_{vv'})$ are, respectively, the edges of H, with $vv' \in E(G-H)$.

Consider now an edge $x_{vv'}u_{n-1}^v$ of G', with $vv' \in E(G-H)$. First suppose that $r(v) \notin \{h_0, h_1\}$. Then $r'(x_{vv'}) = r(v)$. We have already proved that $r'(v)r'(u_{n-1}^v) = r(v)r'(u_{n-1}^v)$ is an edge of H. Hence $r'(x_{vv'})r'(u_{n-1}^v) = r(v)r'(u_{n-1}^v)$ is an edge of H. Now suppose that $r(v) \in \{h_0, h_1\}$. Then $r'(x_{vv'}) = r(v')$. If $r(v) = h_1$ then from our definition of r' we have $r'(u_{n-1}^v) = h_j = h_1 = r(v)$ (as $j = 1 \leq m$), and hence $r'(x_{vv'})r'(u_{n-1}^v) = r(v')r(v)$ is an edge of H. If $r(v) = h_0 (= h_k)$ then from the definition of r' we have $r'(u_{n-1}^v) = h_j = h_0 = r(v)$ (as $j = k \geq k-m+1$), and hence $r'(x_{vv'})r'(u_{n-1}^v) = r(v')r(v)$ is an edge of H. Thus we have proved, under all possibilities, that $r'(x_{vv'})r'(u_{n-1}^v)$ is an edge of H.

Now consider an edge $x_{vv'}w_{n-1}^{v'}$ of G', with $vv' \in E(G-H)$. First suppose that $r(v) \notin \{h_0, h_1\}$. Then $r'(x_{vv'}) = r(v)$. Now first let $r(v') = h_j$, with $1 \leq j \leq n$. Then from our definition of r' we have $r'(w_{n-1}^{v'}) = h_j = r(v')$ if $j > 1$, and $r'(w_{n-1}^{v'}) = h_{j+1} = h_2$ if $j = 1$. Hence if $j > 1$ then $r'(x_{vv'})r'(w_{n-1}^{v'}) = r(v)r(v')$ is an edge of H. If $j = 1$ then since $r(v)$ is adjacent to $r(v') = h_j = h_1$ and $r(v) \notin \{h_0, h_1\}$, it must be that $r(v) = h_2$. Thus if $j = 1$ then $r'(x_{vv'})r'(w_{n-1}^{v'}) = r(v)h_2 = h_2h_2$ is an edge of H. Now let $r(v') = h_j$, with $n+1 \leq j \leq k$. Then from the definition of r' we have $r'(w_{n-1}^{v'}) = h_j = r(v')$ if $j < k$, and $r'(w_{n-1}^{v'}) = h_{j-1} = h_{k-1}$ if $j = k$. Hence if $j < k$ then $r'(x_{vv'})r'(w_{n-1}^{v'}) = r(v)r(v')$ is an edge of H. If $j = k$ then since $r(v)$ is adjacent to $r(v') = h_j = h_k = h_0$ and $r(v) \notin \{h_0, h_1\}$, it must be that $r(v) = h_{k-1}$. Thus if $j = k$ then $r'(x_{vv'})r'(w_{n-1}^{v'}) = r(v)h_{k-1} = h_{k-1}h_{k-1}$ is an edge of H.

Now suppose that $r(v) \in \{h_0, h_1\}$. Then $r'(x_{vv'}) = r(v')$. We have already proved that $r'(v')r'(w_{n-1}^{v'}) = r(v')r'(w_{n-1}^{v'})$ is an edge of H. Hence $r'(x_{vv'})r'(w_{n-1}^{v'}) = r(v')r'(w_{n-1}^{v'})$ is an edge of H. Thus we have proved, under all possibilities, that $r'(x_{vv'})r'(w_{n-1}^{v'})$ is an edge of H.

This completes the proof that $r' : G' \to H$ is a homomorphism. Since $r'(h) = r(h) = h$, for all $h \in V(H)$, it follows that $r' : G' \to H$ is a retraction, and the lemma is proved. □

LEMMA 3.1.2. *G' retracts to H if and only if G' compacts to H.*

Proof. If G' retracts to H then by definition G' compacts to H. Now suppose that $c : G' \to H$ is a compaction. We shall prove that G' retracts to H. We let $U = \{u_1^v | v \in V(G-H)\} \cup \{h_0, h_1\}$ and $W = \{w_1^v | v \in V(G-H)\} \cup \{h_n, h_{n+1}\}$. Note that U and W are cliques in G'.

Since U is a clique in G', $c(U)$ must be a clique in H. Thus $c(U)$ has either one or two vertices. Similarly, $c(W)$ has either one or two vertices. We shall prove that $c(U)$ and $c(W)$ both have two vertices.

Suppose that $c(U)$ has only one vertex. Without loss of generality, let $c(U) = \{h_0\}$ (due to symmetry). We note that $d_{G'}(U, a) < n$ for all $a \in V(G')$. Hence we have $d_{G'}(U, a) < d_H(c(U) = \{h_0\}, h_n) = n$ for all $a \in V(G')$. This implies that $c(a) \neq h_n$ for all $a \in V(G')$. Thus $c(U)$ must have two vertices. We also note that $d_{G'}(W, a) < n$, for all $a \in V(G')$, and hence, similarly, $c(W)$ must also have two vertices.

Thus both $c(U)$ and $c(W)$ are cliques of size two in H. Without loss of generality, suppose that $c(U) = \{h_0, h_1\}$ (due to symmetry). We first prove that $c(W) = \{h_n, h_{n+1}\}$. Let some edge ab of G' cover the edge h_nh_{n+1} of H under c (indeed there exists such an edge in G'). We note that both h_n and h_{n+1} are at distance $n-1$ from $c(U)$ in H. Thus both a and b must be at distance greater than or equal to $n-1$ from U in G'. While there is no vertex at distance greater than $n-1$ from U in G', the only vertices that could possibly be at distance $n-1$ from U in G' are h_n, h_{n+1}, w_1^v, w_{n-1}^v, y_{n-1}^v, v, with $v \in V(G-H)$; and $x_{vv'}$, with $vv' \in E(G-H)$. Thus a and b are among these vertices. Upper bounds on distance to these vertices from U in G' may be obtained due to the following paths in G' (appropriate paths will apply to vertices in question). We are talking of upper bounds, as presence of an edge vh of G, with $v \in V(G-H)$, $h \in V(H)$, may result in a shorter path from U to a vertex mentioned above in G'. The paths are

$h_1h_2 \dots h_{n-1}h_n$, $h_0h_{k-1}h_{k-2} \dots h_{n+2}h_{n+1}$, $u_1^v u_2^v \dots u_m^v a_v w_m^v w_{m-1}^v \dots w_1^v$,
$u_1^v u_2^v \dots u_{m+1}^v w_{m+1}^v w_{m+2}^v \dots w_{n-1}^v$, $u_1^v y_1^v y_2^v \dots y_{n-1}^v$, $u_1^v u_2^v \dots u_{n-1}^v v$,
and $u_1^v u_2^v \dots u_{n-1}^v x_{vv'}$.

Suppose that $c(W) \neq \{h_n, h_{n+1}\}$. Then no edge with both endpoints in W covers

the edge h_nh_{n+1} of H under c, and hence not both a and b belong to W. Thus ab must be an edge among vw^v_{n-1}, $w^v_1y^v_{n-1}$, $y^v_{n-1}h_n$, with $v \in V(G-H)$; vv', $vx_{vv'}$, $v'x_{vv'}$, and $w^{v'}_{n-1}x_{vv'}$, with $vv' \in E(G-H)$; in order to meet the requirements of the edge ab, both vertices in each of these edges are assumed to achieve distance $n-1$ from U in G'. Further, if vh_n or vh_{n+1} is an edge of G, for some vertex $v \in V(G-H)$, then we need to include such an edge also for ab. We shall consider each of these possible edges for ab and show that they do not cover the edge h_nh_{n+1} under c (i.e., $c(\{a,b\}) \neq \{h_n, h_{n+1}\}$), implying that $c(W) = \{h_n, h_{n+1}\}$.

Consider first an edge vw^v_{n-1}, with $v \in V(G-H)$. Since $c(u^v_1) = h_0$ or h_1, either h_n or h_{n+1} is at distance n from $c(u^v_1)$ in H. Since both v and w^v_{n-1} are at distance $n-1$ from u^v_1, it is impossible that $c(\{v, w^v_{n-1}\}) = \{h_n, h_{n+1}\}$. A similar argument applies to an edge $vx_{vv'}$, with $vv' \in E(G-H)$; and an edge $w^v_1y^v_{n-1}$, with $v \in V(G-H)$.

Next consider an edge $y^v_{n-1}h_n$, with $v \in V(G-H)$. If $c(\{y^v_{n-1}, h_n\}) = \{h_n, h_{n+1}\}$ then $c(w^v_1) = h_n$ or h_{n+1}, as w^v_1 is adjacent to y^v_{n-1} and h_n. Since $c(W) \neq \{h_n, h_{n+1}\}$, it must be that $c(w^v_1) = c(h_n)$. This implies that $c(\{y^v_{n-1}, w^v_1\}) = \{h_n, h_{n+1}\}$ which, as noted above, is impossible.

Now consider an edge $vv' \in E(G-H)$. Suppose that $c(\{v,v'\}) = \{h_n, h_{n+1}\}$. Without loss of generality, let $c(v) = h_n$ and $c(v') = h_{n+1}$ (due to symmetry). We have $c(u^v_1), c(u^{v'}_1) \in \{h_0, h_1\}$. Since $d_{G'}(u^v_1, v) = n-1 < d_H(h_0, c(v) = h_n) = n$, this implies that $c(u^v_1) \neq h_0$, and hence $c(u^v_1) = h_1$. Since $d_{G'}(u^{v'}_1, v') = n-1 < d_H(h_1, c(v') = h_{n+1}) = n$, this implies that $c(u^{v'}_1) \neq h_1$, and hence $c(u^{v'}_1) = h_0$. We have the path $u^v_1u^v_2 \ldots u^v_{n-1}v$ of length $n-1$ in G'. Since $c(u^v_1) = h_1$ and $c(v) = h_n$, this implies that $c(u^v_i) = h_i$ for all $i = 1, 2, \ldots, n-1$. Thus we have $c(u^v_{n-1}) = h_{n-1}$. Since $c(w^v_{n-1})$ must be adjacent to $c(v) = h_n$, this implies that $c(w^v_{n-1}) \in \{h_{n-1}, h_n, h_{n+1}\}$. Since $d_{G'}(u^v_1, w^v_{n-1}) = n-1 < d_H(c(u^v_1) = h_1, h_{n+1}) = n$, this implies that $c(w^v_{n-1}) \neq h_{n+1}$, and hence $c(w^v_{n-1}) \in \{h_{n-1}, h_n\}$. Since $c(w^{v'}_{n-1})$ must be adjacent to $c(v') = h_{n+1}$, this implies that $c(w^{v'}_{n-1}) \in \{h_n, h_{n+1}, h_{n+2}\}$. Since $d_{G'}(u^{v'}_1, w^{v'}_{n-1}) = n-1 < d_H(c(u^{v'}_1) = h_0, h_n) = n$, this implies that $c(w^{v'}_{n-1}) \neq h_n$, and hence $c(w^{v'}_{n-1}) \in \{h_{n+1}, h_{n+2}\}$.

We now prove that $c(w^{v'}_{n-1}) \neq h_{n+1}$, which would leave us with the only choice that $c(w^{v'}_{n-1}) = h_{n+2}$. Suppose that $c(w^{v'}_{n-1}) = h_{n+1}$. We have the path $u^{v'}_1u^{v'}_2 \ldots u^{v'}_{m+1}w^{v'}_{m+1}w^{v'}_{m+2} \ldots w^{v'}_{n-1}$ of length $n-1$ in G'. Since $c(u^{v'}_1) = h_0$ and $c(w^{v'}_{n-1}) = h_{n+1}$, this implies that $c(u^{v'}_i) = h_{k-i+1}$ for all $i = 1, 2, \ldots, m+1$, and $c(w^{v'}_i) = h_{k-i}$ for all $i = m+1, m+2, \ldots, n-1$. Thus we have $c(u^{v'}_m) = h_{k-m+1}$ and $c(w^{v'}_{m+1}) = h_{k-m-1}$. Since $w^{v'}_m$ is adjacent to $w^{v'}_{m+1}$ and at distance two from $u^{v'}_m$ in G', this implies that $c(w^{v'}_m) = h_{k-m-1}$ or h_{k-m}. Since $w^{v'}_1$ is at distance $m-1$ from $w^{v'}_m$ in G', it follows that $c(w^{v'}_1) = h_s$, $n+1 \leq s \leq k-1$. We also have the path $u^v_1u^v_2 \ldots u^v_{m+1}w^v_{m+1}w^v_{m+2} \ldots w^v_{n-1}$ of length $n-1$ in G'. As shown above, we have $c(u^v_i) = h_i$, for all $i = 1, 2, \ldots, m+1$, and $c(w^v_{n-1}) = h_n$ or h_{n-1}. This implies that $c(w^v_i) = h_{i+1}$ or h_i for all $i = m+1, m+2, \ldots, n-1$. Thus we have $c(u^v_m) = h_m$ and $c(w^v_{m+1}) = h_{m+1}$ or h_{m+2}. Since w^v_m is adjacent to w^v_{m+1} and at distance two from u^v_m in G', this implies that $c(w^v_m) = h_m, h_{m+1}$, or h_{m+2}. Since w^v_1 is at distance $m-1$ from w^v_m in G', it follows that $c(w^v_1) = h_t$, $1 \leq t \leq n$. We have that $c(w^v_1)$ and $c(w^{v'}_1)$ must be adjacent in H. The only possible pair of values for t and s are n and $n+1$, respectively, for $c(w^v_1)$ to be adjacent to $c(w^{v'}_1)$. Thus $c(w^v_1) = h_n$ and $c(w^{v'}_1) = h_{n+1}$, which implies that $c(W) = \{h_n, h_{n+1}\}$ and we have a contradiction. Thus $c(w^{v'}_{n-1}) \neq h_{n+1}$, and hence $c(w^{v'}_{n-1}) = h_{n+2}$.

Now, $c(x_{vv'})$ must be adjacent to $c(u^v_{n-1}) = h_{n-1}$, $c(v) = h_n$, $c(v') = h_{n+1}$, and $c(w^{v'}_{n-1}) = h_{n+2}$, which is impossible.

Consider now an edge $v'x_{vv'}$, with $vv' \in E(G-H)$. If $c(\{v', x_{vv'}\}) = \{h_n, h_{n+1}\}$ then $c(v) = h_n$ or h_{n+1}, as v is adjacent to v' and $x_{vv'}$. This implies that either $c(\{v, v'\}) = \{h_n, h_{n+1}\}$ or $c(\{v, x_{vv'}\}) = \{h_n, h_{n+1}\}$, both of which we have already proved do not hold.

Now consider an edge $w^{v'}_{n-1}x_{vv'}$, with $vv' \in E(G-H)$. If $c(\{w^{v'}_{n-1}, x_{vv'}\}) = \{h_n, h_{n+1}\}$ then $c(v') = h_n$ or h_{n+1}, as v' is adjacent to $w^{v'}_{n-1}$ and $x_{vv'}$. This implies that either $c(\{v', w^{v'}_{n-1}\}) = \{h_n, h_{n+1}\}$ or $c(\{v', x_{vv'}\}) = \{h_n, h_{n+1}\}$, both of which we have already proved do not hold.

Finally, consider a possible edge vh_n, with $v \in V(G-H)$. Suppose that $c(\{v, h_n\}) = \{h_n, h_{n+1}\}$. Without loss of generality, let $c(v) = h_n$ and $c(h_n) = h_{n+1}$ (due to symmetry). Exactly, as in the case $vv' \in E(G-H)$ above, we establish that $c(w^v_1) = h_t$, $1 \le t \le n$. Since $c(w^v_1)$ must be adjacent to $c(h_n) = h_{n+1}$ in H, it follows that $c(w^v_1) = h_n$, and hence $h_n, h_{n+1} \in c(W)$, which is a contradiction. We argue similarly for a possible edge vh_{n+1}, with $v \in V(G-H)$.

This completes the proof that $c(W) = \{h_n, h_{n+1}\}$. We now prove that $c(h_0) \neq c(h_1)$. Suppose to the contrary that $c(h_0) = c(h_1)$. We have $c(h_0), c(h_1) \in \{h_0, h_1\}$, as $c(U) = \{h_0, h_1\}$. Without loss of generality, let $c(h_0) = c(h_1) = h_0$ (due to symmetry). Since $c(U) = \{h_0, h_1\}$, $c(u_v) = h_1$ for some vertex v of $G-H$. We have $c(w^v_1), c(h_n) \in \{h_n, h_{n+1}\}$, as $c(W) = \{h_n, h_{n+1}\}$. Since $d_{G'}(h_1, h_n) = n-1 < d_H(c(h_1) = h_0, h_n) = n$, this implies that $c(h_n) \neq h_n$, and hence $c(h_n) = h_{n+1}$. Since $d_{G'}(u^v_1, w^v_1) = n-1 < d_H(c(u^v_1) = h_1, h_{n+1}) = n$, this implies that $c(w^v_1) \neq h_{n+1}$, and hence $c(w^v_1) = h_n$. Now, $c(y^v_{n-1})$ must be adjacent to $c(h_n) = h_{n+1}$ and $c(w^v_1) = h_n$. Hence $c(y^v_{n-1}) \in \{h_n, h_{n+1}\}$. Also, $c(y^v_1)$ must be adjacent to $c(h_0) = h_0$ and $c(u^v_1) = h_1$. Hence $c(y^v_1) \in \{h_0, h_1\}$. We have $d_{G'}(y^v_1, y^v_{n-1}) = n-2 < d_H(\{h_0, h_1\}, \{h_n, h_{n+1}\}) = n-1$. Hence it is impossible that $c(y^v_1) \in \{h_0, h_1\}$ and $c(y^v_{n-1}) \in \{h_n, h_{n+1}\}$.

Thus $c(h_0) \neq c(h_1)$, i.e., $c(\{h_0, h_1\}) = \{h_0, h_1\}$. Without loss of generality, suppose that $c(h_0) = h_0$ and $c(h_1) = h_1$ (due to symmetry). We have $c(h_n), c(h_{n+1}) \in \{h_n, h_{n+1}\}$, as $c(W) = \{h_n, h_{n+1}\}$. Since $d_{G'}(h_1, h_n) = n-1 < d_H(c(h_1) = h_1, h_{n+1}) = n$, this implies that $c(h_n) \neq h_{n+1}$, and hence $c(h_n) = h_n$. Since $d_{G'}(h_0, h_{n+1}) = n-1 < d_H(c(h_0) = h_0, h_n) = n$, this implies that $c(h_{n+1}) \neq h_n$, and hence $c(h_{n+1}) = h_{n+1}$. We have the path $h_1h_2 \dots h_n$ of length $n-1$ in G'. Since $c(h_1) = h_1$ and $c(h_n) = h_n$, this implies that $c(h_i) = h_i$ for all $i = 1, 2, 3, \dots, n$. We also have the path $h_0h_{k-1}h_{k-2} \dots h_{n+1}$ of length $n-1$ in G'. Since $c(h_0) = h_0$ and $c(h_{n+1}) = h_{n+1}$, this implies that $c(h_i) = h_i$ for all $i = 0, n+1, n+2, n+3, \dots, k-1$. Thus we have $c(h_i) = h_i$ for all $i = 0, 1, 2, \dots, k-1$. Hence $c : G' \to H$ is a retraction and the lemma is proved. □

Proof of Theorem 3.1 in the case $k = 4m$. The construction of G' is as follows. For each vertex v in $V(G-H)$, we add to G three vertex disjoint paths U_v, W_v, and Y_v, each containing $n-1$ new vertices. Let $U_v = u^v_1u^v_2 \dots u^v_{n-1}$, $W_v = w^v_1w^v_2 \dots w^v_{n-1}$, and $Y_v = y^v_1y^v_2 \dots y^v_{n-1}$, with $v \in V(G-H)$. We add the edges $u^v_1h_0$, $u^v_1h_1$, vu^v_{n-1}, $w^v_1h_n$, $w^v_1h_{n+1}$, vw^v_{n-1}, $u^v_mw^v_m$, $y^v_1h_0$, $y^v_{n-1}h_n$, $u^v_1y^v_1$, and $w^v_1y^v_{n-1}$, with $v \in V(G-H)$. For convenience, in the proof of the first lemma, as for the case $k = 4m+2$, we also use h_k to denote the vertex h_0. See Figure 3.4. Furthermore, we add the edges $u^v_1u^{v'}_1$ and $w^v_1w^{v'}_1$ for all $v, v' \in V(G-H)$, $v \neq v'$.

The addition of a new vertex $x_{vv'}$ and the edges associated with it are same as for the case $k = 4m+2$, with $vv' \in E(G-H)$, $v \neq v'$. See Figure 3.3. This completes

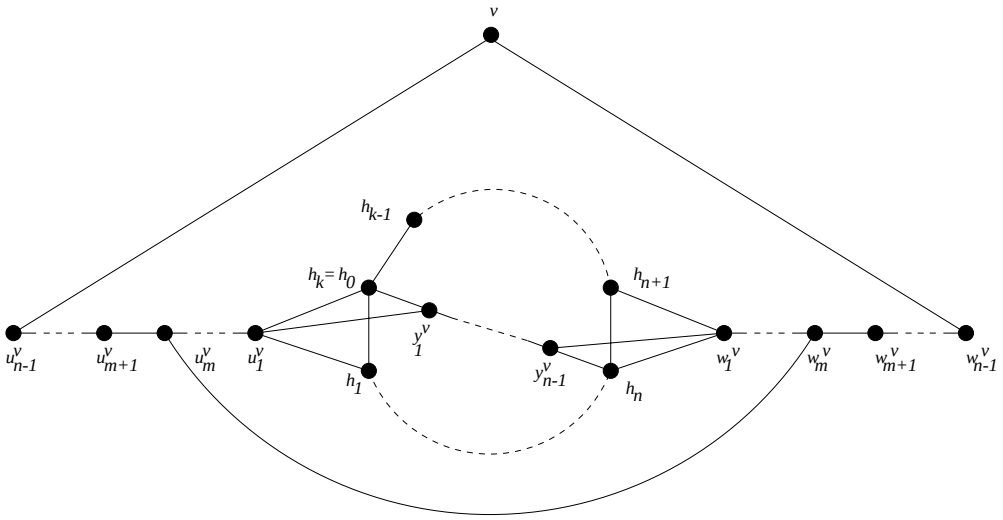


FIG. 3.4. *Construction of G' for a vertex v in $G - H$, with $k = 4m$.*

the construction of G'.

Note that the construction of G' is like for $k = 4m+2$ except that now we do not have the vertex a_v and the edge $u^v_{m+1}w^v_{m+1}$, with $v \in V(G-H)$. Instead we have the edge $u^v_m w^v_m$, with $v \in V(G-H)$. The two lemmas that we gave for the case when $k = 4m+2$ hold similarly for the case when $k = 4m$ with some variations in their proofs as noted below.

LEMMA 3.1.3. *G retracts to H if and only if G' retracts to H.*

Proof. If G' retracts to H then clearly G retracts to H, as G is a subgraph of G'. If $r : G \to H$ is a retraction then we define a retraction $r' : G' \to H$ exactly as we defined in the case $k = 4m+2$ in Lemma 3.1.1 except that now we do not need to define $r'(a_v)$, with $v \in V(G-H)$. As for the case $k = 4m+2$, it can be verified that for every edge ab of G', $r'(a)r'(b)$ is indeed an edge of H. Recall that the edges of G' are ab, $u^v_i u^v_{i+1}$, $w^v_i w^v_{i+1}$, $y^v_i y^v_{i+1}$, $u^v_1 h_0$, $u^v_1 h_1$, $w^v_1 h_n$, $w^v_1 h_{n+1}$, $v u^v_{n-1}$, $v w^v_{n-1}$, $u^v_m w^v_m$, $y^v_1 h_0$, $y^v_{n-1} h_n$, $u^v_1 y^v_1$, $w^v_1 y^v_{n-1}$, $u^v_1 u^{v'}_1$, $w^v_1 w^{v'}_1$, $g x_{gg'}$, $g' x_{gg'}$, $x_{gg'} u^g_{n-1}$, and $x_{gg'} w^{g'}_{n-1}$, with $ab \in E(G)$, $v, v' \in V(G-H)$, $gg' \in E(G-H)$, $i = 1, 2, \ldots, n-2$. □

LEMMA 3.1.4. *G' retracts to H if and only if G' compacts to H.*

Proof. If G' retracts to H then by definition G' compacts to H. Now suppose that $c : G' \to H$ is a compaction. We shall prove that G' retracts to H. We define cliques U and W, as we did for the case $k = 4m+2$ in Lemma 3.1.2, i.e., we let $U = \{u^v_1 | v \in V(G-H)\} \cup \{h_0, h_1\}$ and $W = \{w^v_1 | v \in V(G-H)\} \cup \{h_n, h_{n+1}\}$.

We note that $d_{G'}(U, a) < n$ and $d_{G'}(W, a) < n$ for all $a \in V(G')$. Thus, exactly as for the case $k = 4m+2$, we establish that both $c(U)$ and $c(W)$ are cliques of size two in H. Without loss of generality, suppose that $c(U) = \{h_0, h_1\}$ (due to symmetry). We first prove that $c(W) = \{h_n, h_{n+1}\}$. Let some edge ab of G' cover the edge $h_n h_{n+1}$ of H under c. As for the case $k = 4m+2$, we have that both a and b are at distance $n-1$ from U in G'. The only vertices that could possibly be at distance $n-1$ from U in G' are h_n, h_{n+1}, w^v_1, w^v_{n-1}, y^v_{n-1}, v, with $v \in V(G-H)$; and $x_{vv'}$, with $vv' \in E(G-H)$. Upper bounds on distance to these vertices from U in G' may be obtained due to the following paths in G':

$h_1h_2\dots h_{n-1}h_n$, $h_0h_{k-1}h_{k-2}\dots h_{n+2}h_{n+1}$, $u_1^vu_2^v\dots u_m^vw_m^vw_{m-1}^v\dots w_1^v$, $u_1^vu_2^v\dots u_m^vw_m^vw_{m+1}^v\dots w_{n-1}^v$, $u_1^vy_1^vy_2^v\dots y_{n-1}^v$, $u_1^vu_2^v\dots u_{n-1}^vv$, and $u_1^vu_2^v\dots u_{n-1}^vx_{vv'}$.

Notice that the paths to w_1^v and w_{n-1}^v from u_1^v, mentioned above, are different from the case $k=4m+2$.

Suppose that $c(W)\neq\{h_n,h_{n+1}\}$. Then, just as for the case $k=4m+2$, we conclude that ab must be an edge among vw_{n-1}^v, $w_1^vy_{n-1}^v$, $y_{n-1}^vh_n$, with $v\in V(G-H)$; vv', $vx_{vv'}$, $v'x_{vv'}$, and $w_{n-1}^{v'}x_{vv'}$, with $vv'\in E(G-H)$; in order to meet the requirements of the edge ab, both vertices in each of these edges are assumed to achieve distance $n-1$ from U in G'. Further, if vh_n or vh_{n+1} is an edge of G, for some vertex $v\in V(G-H)$, then we need to include such an edge also for ab. Note that the set of these possible edges for ab is the same as for the case $k=4m+2$. For all these possible edges of ab except $vv'\in E(G-H)$, the proof that these edges do not cover the edge h_nh_{n+1} under c (i.e., $c(\{a,b\})\neq\{h_n,h_{n+1}\}$) is exactly the same as for the case $k=4m+2$.

For an edge vv', we now make use of the paths $u_1^vu_2^v\dots u_m^vw_m^vw_{m+1}^v\dots w_{n-1}^v$ and $u_1^{v'}u_2^{v'}\dots u_m^{v'}w_m^{v'}w_{m+1}^{v'}\dots w_{n-1}^{v'}$ instead of the paths $u_1^vu_2^v\dots u_{m+1}^vw_{m+1}^vw_{m+2}^v\dots w_{n-1}^v$ and $u_1^{v'}u_2^{v'}\dots u_{m+1}^{v'}w_{m+1}^{v'}w_{m+2}^{v'}\dots w_{n-1}^{v'}$, respectively, with $vv'\in E(G-H)$. We consider this case and show its proof.

Let vv' be an edge of $G-H$. Suppose that $c(\{v,v'\})=\{h_n,h_{n+1}\}$. Without loss of generality, let $c(v)=h_n$ and $c(v')=h_{n+1}$ (due to symmetry). Exactly, as in the case $k=4m+2$, we prove that $c(u_i^v)=h_i$ for all $i=1,2,\dots,n-1$, $c(u_1^{v'})=h_0$, $c(w_{n-1}^v)\in\{h_{n-1},h_n\}$, and $c(w_{n-1}^{v'})\in\{h_{n+1},h_{n+2}\}$.

Now we prove that $c(w_{n-1}^{v'})\neq h_{n+1}$, which would leave us with the only choice that $c(w_{n-1}^{v'})=h_{n+2}$. Suppose that $c(w_{n-1}^{v'})=h_{n+1}$. We have the path $u_1^{v'}u_2^{v'}\dots u_m^{v'}w_m^{v'}w_{m+1}^{v'}\dots w_{n-1}^{v'}$ of length $n-1$ in G'. Since $c(u_1^{v'})=h_0$ and $c(w_{n-1}^{v'})=h_{n+1}$, this implies that $c(u_i^{v'})=h_{k-i+1}$ for all $i=1,2,\dots,m$, and $c(w_i^{v'})=h_{k-i}$ for all $i=m,m+1,\dots,n-1$. Thus we have $c(w_m^{v'})=h_{k-m}$. Since $w_1^{v'}$ is at distance $m-1$ from $w_m^{v'}$ in G', it follows that $c(w_1^{v'})=h_s$, $n+1\leq s\leq k-1$. We also have the path $u_1^vu_2^v\dots u_m^vw_m^vw_{m+1}^v\dots w_{n-1}^v$ of length $n-1$ in G'. Since $c(u_i^v)=h_i$, for all $i=1,2,\dots,m$, and $c(w_{n-1}^v)=h_n$ or h_{n-1}, this implies that $c(w_i^v)=h_{i+1}$ or h_i for all $i=m,m+1,\dots,n-1$. Thus we have $c(w_m^v)=h_m$ or h_{m+1}. Since w_1^v is at distance $m-1$ from w_m^v in G', it follows that $c(w_1^v)=h_t$, $1\leq t\leq n$. We have that $c(w_1^v)$ and $c(w_1^{v'})$ must be adjacent in H. The only possible pair of values for t and s are n and $n+1$, respectively, for $c(w_1^v)$ to be adjacent to $c(w_1^{v'})$. Thus $c(w_1^v)=h_n$ and $c(w_1^{v'})=h_{n+1}$, which implies that $c(W)=\{h_n,h_{n+1}\}$ and we have a contradiction. Thus $c(w_{n-1}^{v'})\neq h_{n+1}$, and hence $c(w_{n-1}^{v'})=h_{n+2}$.

Now, $c(x_{vv'})$ must be adjacent to $c(u_{n-1}^v)=h_{n-1}$, $c(v)=h_n$, $c(v')=h_{n+1}$, and $c(w_{n-1}^{v'})=h_{n+2}$, which is impossible. Hence $c(\{v,v'\})\neq\{h_n,h_{n+1}\}$.

Thus we have that $c(W)=\{h_n,h_{n+1}\}$. Exactly, as for the case $k=4m+2$, we prove that $c(h_0)\neq c(h_1)$. Without loss of generality, we let $c(h_0)=h_0$ and $c(h_1)=h_1$ (due to symmetry), and exactly as for the case $k=4m+2$, we establish that $c(h_i)=h_i$ for all $i=0,1,2,\dots,k-1$, i.e., $c:G'\to H$ is a retraction. □

Proof of Theorem 3.1 in the case $k=4m+3$. The construction of G' is as follows. For each vertex v in $G-H$, we add to G three vertex disjoint paths U_v, W_v, and Y_v, where U_v and W_v each contains $n-1$ new vertices, and Y_v contains n new vertices. Let $U_v=u_1^vu_2^v\dots u_{n-1}^v$, $W_v=w_1^vw_2^v\dots w_{n-1}^v$, and $Y_v=y_1^vy_2^v\dots y_n^v$,

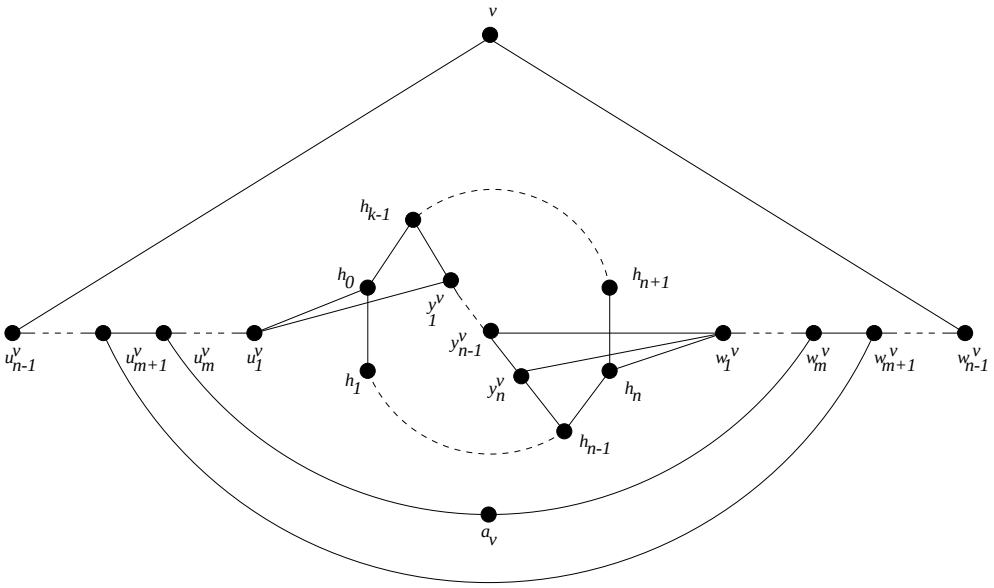


FIG. 3.5. *Construction of G' for a vertex v in $G - H$, with $k = 4m + 3$.*

with $v \in V(G - H)$. We add the edges $h_0u_1^v$, vu_{n-1}^v, $h_nw_1^v$, vw_{n-1}^v, $u_{m+1}^v w_{m+1}^v$, $h_{k-1}y_1^v$, $h_{n-1}y_n^v$, $u_1^v y_1^v$, $w_1^v y_n^v$, and $w_1^v y_{n-1}^v$, with $v \in V(G - H)$ (note that h_n is a vertex opposite to h_0 in H, and h_{n-1} is a vertex opposite to h_{k-1} in H). For each $v \in V(G - H)$, we also add a new vertex a_v adjacent to u_m^v and w_m^v. See Figure 3.5.

The addition of a new vertex $x_{vv'}$ and the edges associated with it are same as for the case $k = 4m + 2$, with $vv' \in E(G - H)$, $v \neq v'$. See Figure 3.3.

This completes the construction of G'. As for the case $k = 4m + 2$, we now prove the following two lemmas in order to prove the theorem for $k = 4m + 3$.

LEMMA 3.1.5. *G retracts to H if and only if G' retracts to H.*

Proof. If G' retracts to H then clearly G retracts to H, as G is a subgraph of G'. Now suppose that $r : G \to H$ is a retraction. Below, we define a retraction $r' : G' \to H$. Similar to the case $k = 4m + 2$, it can be verified that for the edges ab of G', $r'(a)r'(b)$ is indeed an edge of H. Recall that the edges of G' are (note the difference in the set of edges as compared to the case $k = 4m+2$) ab, $u_i^v u_{i+1}^v$, $w_i^v w_{i+1}^v$, $y_j^v y_{j+1}^v$, $u_1^v h_0$, $w_1^v h_n$, vu_{n-1}^v, vw_{n-1}^v, $u_{m+1}^v w_{m+1}^v$, $a_v u_m^v$, $a_v w_m^v$, $y_1^v h_{k-1}$, $y_n^v h_{n-1}$, $u_1^v y_1^v$, $w_1^v y_{n-1}^v$, $w_1^v y_n^v$, $gx_{gg'}$, $g'x_{gg'}$, $x_{gg'}u_{n-1}^g$, and $x_{gg'}w_{n-1}^{g'}$, with $ab \in E(G)$, $v \in V(G-H)$, $gg' \in E(G-H)$, $i = 1, 2, \ldots, n-2$, $j = 1, 2, \ldots, n-1$. We shall, however, include the verification for the edges $x_{vv'}u_{n-1}^v$ and $x_{vv'}w_{n-1}^{v'}$, with $vv' \in E(G - H)$, as it may be helpful.

For each vertex v of the graph G, we define

$$r'(v) = r(v).$$

Now we fix a vertex $v \in V(G - H)$ for defining r' for the vertices of U_v, W_v, and Y_v, and for the vertex a_v. Let $r(v) = h_j$. We shall define r' for the said vertices when $0 \leq j \leq n$, and when $n + 1 \leq j \leq k - 1$.

First assume that $0 \leq j \leq n$.
For the vertices of U_v, we define r' as follows.
If $j = 0$ then we define
$r'(u_i^v) = h_{i-1}$ for all $i = 1, 2, \ldots, m$,
$r'(u_i^v) = h_{n-i-1}$ for all $i = m+1, m+2, \ldots, n-1$.
If $1 \leq j \leq m$ then we define
$r'(u_i^v) = h_i$ for all $i = 1, 2, \ldots, m$,
$r'(u_i^v) = h_{n-i}$ for all $i = m+1, m+2, \ldots, n-j$,
$r'(u_i^v) = h_j$ for all $i = n-j+1, n-j+2, \ldots, n-1$.
If $j > m$ then we define
$r'(u_i^v) = h_i$ for all $i = 1, 2, \ldots, j-1$,
$r'(u_i^v) = h_j$ for all $i = j, j+1, \ldots, n-1$.
For the vertices of W_v, we define r' as follows.
If $j = n$ then we define
$r'(w_i^v) = h_{n-i}$ for all $i = 1, 2, \ldots, m-1$,
$r'(w_i^v) = h_{i+1}$ for all $i = m, m+1, m+2, \ldots, n-1$.
If $n - m \leq j \leq n-1$ then we define
$r'(w_i^v) = h_{n-i}$ for all $i = 1, 2, \ldots, m$,
$r'(w_i^v) = h_i$ for all $i = m+1, m+2, \ldots, j$,
$r'(w_i^v) = h_j$ for all $i = j+1, j+2, \ldots, n-1$.
If $j < n - m$ then we define
$r'(w_i^v) = h_{n-i}$ for all $i = 1, 2, \ldots, n-j-1$,
$r'(w_i^v) = h_j$ for all $i = n-j, n-j+1, \ldots, n-1$.
For the vertex a_v, we define r' as follows.
If $j = 0$ then we define
$r'(a_v) = h_m$.
If $j \neq 0$ then we define
$r'(a_v) = h_{m+1}$.
For the vertices of Y_v, we define
$r'(y_i^v) = h_{i-1}$ for all $i = 1, 2, \ldots, n$.
Now assume that $n + 1 \leq j \leq k - 1$.
For the vertices of U_v, we define r' as follows.
If $j \geq k - m$ then we define
$r'(u_i^v) = h_{k-i}$ for all $i = 1, 2, \ldots, m$,
$r'(u_i^v) = h_{n+i+1}$ for all $i = m+1, m+2, \ldots, j-n-1$,
$r'(u_i^v) = h_j$ for all $i = j-n, j-n+1, \ldots, n-1$.
If $j < k - m$ then we define
$r'(u_i^v) = h_{k-i}$ for all $i = 1, 2, \ldots, k-j-1$,
$r'(u_i^v) = h_j$ for all $i = k-j, k-j+1, \ldots, n-1$.
For the vertices of W_v, we define r' as follows.
If $j \leq n + m$ then we define
$r'(w_i^v) = h_{n+i}$ for all $i = 1, 2, \ldots, m$,
$r'(w_i^v) = h_{k-i-1}$ for all $i = m+1, m+2, \ldots, k-j-1$,
$r'(w_i^v) = h_j$ for all $i = k-j, k-j+1, \ldots, n-1$.
If $j > n + m$ then we define
$r'(w_i^v) = h_{n+i}$ for all $i = 1, 2, \ldots, j-n-1$,
$r'(w_i^v) = h_j$ for all $i = j-n, j-n+1, \ldots, n-1$.
For the vertex a_v, we define
$r'(a_v) = h_{k-m-1}$.

For the vertices of Y_v, we define

$r'(y_i^v) = h_{k-i-1}$ for all $i = 1, 2, \ldots, n$.

This completes the definition of r' for the vertices of U_v, W_v, and Y_v, and for the vertex a_v.

For the vertex $x_{vv'}$, with $vv' \in E(G - H)$, we define

$r'(x_{vv'}) = r(v)$, if $r(v) \notin \{h_0, h_{k-1}\}$, and

$r'(x_{vv'}) = r(v')$, if $r(v) \in \{h_0, h_{k-1}\}$.

We now consider the edges $x_{vv'}u_{n-1}^v$ and $x_{vv'}w_{n-1}^{v'}$ of G' and show that $r'(x_{vv'})r'(u_{n-1}^v)$ and $r'(x_{vv'})r'(w_{n-1}^{v'})$ are the edges of H, with $vv' \in E(G - H)$. It may be helpful to include the proof for these edges.

Consider first an edge $x_{vv'}u_{n-1}^v$ of G', with $vv' \in E(G - H)$. First suppose that $r(v) \notin \{h_0, h_{k-1}\}$. Then $r'(x_{vv'}) = r(v)$. We already have that $r'(v)r'(u_{n-1}^v) = r(v)r'(u_{n-1}^v)$ is an edge of H. Hence $r'(x_{vv'})r'(u_{n-1}^v) = r(v)r'(u_{n-1}^v)$ is an edge of H. Now suppose that $r(v) \in \{h_0, h_{k-1}\}$. Then $r'(x_{vv'}) = r(v')$. If $r(v) = h_0$ then from our definition of r' we have $r'(u_{n-1}^v) = h_0 = r(v)$ (as $j = 0$), and hence $r'(x_{vv'})r'(u_{n-1}^v) = r(v')r(v)$ is an edge of H. If $r(v) = h_{k-1}$ then from our definition of r' we have $r'(u_{n-1}^v) = h_j = h_{k-1} = r(v)$ (as $j = k - 1 \geq k - m$), and hence $r'(x_{vv'})r'(u_{n-1}^v) = r(v')r(v)$ is an edge of H. Thus we have proved, under all possibilities, that $r'(x_{vv'})r'(u_{n-1}^v)$ is an edge of H.

Now consider an edge $x_{vv'}w_{n-1}^{v'}$, with $vv' \in E(G-H)$. First suppose that $r(v) \notin \{h_0, h_{k-1}\}$. Then $r'(x_{vv'}) = r(v)$. Now first let $r(v') = h_j$, with $0 \leq j \leq n$. Then from our definition of r' we have $r'(w_{n-1}^{v'}) = h_j = r(v')$, if $j > 0$, and $r'(w_{n-1}^{v'}) = h_{j+1} = h_1$ if $j = 0$. Hence if $j > 0$ then $r'(x_{vv'})r'(w_{n-1}^{v'}) = r(v)r(v')$ is an edge of H. If $j = 0$ then since $r(v)$ is adjacent to $r(v') = h_j = h_0$ and $r(v) \notin \{h_0, h_{k-1}\}$, it must be that $r(v) = h_1$. Thus if $j = 0$ then $r'(x_{vv'})r'(w_{n-1}^{v'}) = r(v)h_1 = h_1h_1$ is an edge of H. Now let $r(v') = h_j$, with $n + 1 \leq j \leq k - 1$. Then from our definition of r' we have $r'(w_{n-1}^{v'}) = h_j = r(v')$, if $j < k - 1$, and $r'(w_{n-1}^{v'}) = h_{j-1} = h_{k-2}$, if $j = k - 1$. Hence if $j < k - 1$ then $r'(x_{vv'})r'(w_{n-1}^{v'}) = r(v)r(v')$ is an edge of H. If $j = k - 1$ then since $r(v)$ is adjacent to $r(v') = h_j = h_{k-1}$ and $r(v) \notin \{h_0, h_{k-1}\}$, it must be that $r(v) = h_{k-2}$. Thus if $j = k - 1$ then $r'(x_{vv'})r'(w_{n-1}^{v'}) = r(v)h_{k-2} = h_{k-2}h_{k-2}$ is an edge of H.

Now suppose that $r(v) \in \{h_0, h_{k-1}\}$. Then $r'(x_{vv'}) = r(v')$. We already have that $r'(v')r'(w_{n-1}^{v'}) = r(v')r'(w_{n-1}^{v'})$ is an edge of H. Hence $r'(x_{vv'})r'(w_{n-1}^{v'}) = r(v')r'(w_{n-1}^{v'})$ is an edge of H. Thus we have proved, under all possibilities, that $r'(x_{vv'})r'(w_{n-1}^{v'})$ is an edge of H.

Thus, as for $k = 4m + 2$, we conclude that $r' : G' \to H$ is a homomorphism and a retraction. □

LEMMA 3.1.6. *G' retracts to H if and only if G' compacts to H.*

Proof. If G' retracts to H then by definition G' compacts to H. Now suppose that $c : G' \to H$ is a compaction. We shall prove that G' retracts to H. We define the sets $U = \{u_1^v | v \in V(G - H)\} \cup \{h_1, h_0, h_{k-1}\}$ and $W = \{w_1^v | v \in V(G - H)\} \cup \{h_{n-1}, h_n, h_{n+1}\}$ (unlike the even k case, U and W are not cliques in G').

Since the subgraph of G' induced by the vertices in U is of diameter two (with h_0 adjacent to every vertex in U), the vertices of $c(U)$ induce a path of length at most two in H. Thus $c(U)$ has either one, two, or three vertices. Similarly, $c(W)$ has either one, two, or three vertices. We shall prove that both $c(U)$ and $c(W)$ have three vertices.

Suppose that $c(U)$ has only one or two vertices, i.e., $c(U)$ induces a path of length

at most one in H. Without loss of generality, let $c(U) = \{h_0\}$ or $c(U) = \{h_0, h_1\}$ (due to symmetry). We note that $d_{G'}(U, a) < n$ for all $a \in V(G')$. Hence we have $d_{G'}(U, a) < d_H(c(U), h_{n+1}) = n$ for all $a \in V(G')$. This implies that $c(a) \neq h_{n+1}$ for all $a \in V(G')$. Thus $c(U)$ must have three vertices. We also note that $d_{G'}(W, a) < n$, for all $a \in V(G')$, and hence, similarly, $c(W)$ must also have three vertices. We point out here that upper bounds on $d_{G'}(U, y)$ and $d_{G'}(W, y)$, for the vertices y of Y_v, with $v \in V(G - H)$, are obtained differently due to the following paths in G':
$u_1^v y_1^v y_2^v \dots y_{n-1}^v$, $h_1 h_2 \dots h_{n-1} y_n^v$, $w_1^v y_{n-1}^v y_{n-2}^v \dots y_1^v$, and $w_1^v y_n^v$.

Thus $c(U)$ and $c(W)$ both induce paths containing three vertices in H. Without loss of generality, suppose that $c(U) = \{h_1, h_0, h_{k-1}\}$ (due to symmetry). We first prove that $\{h_n, h_{n+1}\} \subset c(W)$, i.e., $c(W) = \{h_{n-1}, h_n, h_{n+1}\}$ or $c(W) = \{h_n, h_{n+1}, h_{n+2}\}$. Let some edge ab of G' cover the edge $h_n h_{n+1}$ of H under c (clearly there exists such an edge in G'). We note that both h_n and h_{n+1} are at distance $n-1$ from $c(U)$ in H. Thus both a and b must be at distance greater than or equal to $n-1$ from U in G'. While there is no vertex at distance greater than $n-1$ from U in G', the only vertices that could possibly be at distance $n-1$ from U in G' are h_n, h_{n+1}, w_1^v, w_{n-1}^v, y_{n-1}^v, y_n^v, v, with $v \in V(G-H)$; and $x_{vv'}$, with $vv' \in E(G-H)$. Upper bounds on distance to these vertices from U in G' may be obtained due to the following paths in G' (appropriate paths will apply to vertices in question). We are mentioning upper bounds, as presence of an edge vh of G, with $v \in V(G - H)$, $h \in V(H)$, may result in a shorter path from U to a vertex mentioned above in G'. The paths are
$h_1 h_2 \dots h_n$, $h_{k-1} h_{k-2} \dots h_{n+1}$, $u_1^v u_2^v \dots u_m^v a_v w_m^v w_{m-1}^v \dots w_1^v$,
$u_1^v u_2^v \dots u_{m+1}^v w_{m+1}^v w_{m+2}^v \dots w_{n-1}^v$, $u_1^v y_1^v y_2^v \dots y_{n-1}^v$, $h_1 h_2 \dots h_{n-1} y_n^v$,
$u_1^v u_2^v \dots u_{n-1}^v v$, and $u_1^v u_2^v \dots u_{n-1}^v x_{vv'}$.

Suppose that $\{h_n, h_{n+1}\} \not\subset c(W)$. Then no edge with both endpoints in W covers the edge $h_n h_{n+1}$ of H under c, and hence not both a and b belong to W. Thus ab must be an edge among vw_{n-1}^v, $w_1^v y_n^v$, $w_1^v y_{n-1}^v$, $y_{n-1}^v y_n^v$, with $v \in V(G-H)$; vv', $vx_{vv'}$, $v'x_{vv'}$, and $w_{n-1}^{v'} x_{vv'}$, with $vv' \in E(G-H)$; in order to meet the requirements of the edge ab, both vertices in each of these edges are assumed to achieve distance $n-1$ from U in G'. Further, if vh_n or vh_{n+1} is an edge of G, for some vertex $v \in V(G-H)$, then we need to include such an edge also for ab. We shall consider each of these possible edges for ab and show that they do not cover the edge $h_n h_{n+1}$ under c (i.e., $c(\{a, b\}) \neq \{h_n, h_{n+1}\}$), implying that $\{h_n, h_{n+1}\} \subset c(W)$.

For the edges vw_{n-1}^v, $w_1^v y_{n-1}^v$, with $v \in V(G-H)$; and $vx_{vv'}$, with $vv' \in E(G-H)$, we argue exactly as in the case $k = 4m+2$ except that now $c(u_1^v) = h_{k-1}, h_0$, or h_1.

Consider now an edge $y_n^v w_1^v$, with $v \in V(G-H)$. Suppose that $c(\{y_n^v, w_1^v\}) = \{h_n, h_{n+1}\}$. Without loss of generality, let $c(y_n^v) = h_n$ and $c(w_1^v) = h_{n+1}$ (due to symmetry). We have $c(h_1) = h_1, h_0$, or h_{k-1}. Since $d_{G'}(h_1, y_n^v) = n-1 < d_H(\{h_0, h_{k-1}\}, c(y_n^v) = h_n) = n$, this implies that $c(h_1) \notin \{h_0, h_{k-1}\}$, and hence $c(h_1) = h_1$. We have the path $h_1 h_2 \dots h_{n-1} y_n^v$ of length $n-1$ in G'. Since $c(h_1) = h_1$ and $c(y_n^v) = h_n$, this implies that $c(h_i) = h_i$ for all $i = 1, 2, \dots, n-1$. Since $c(h_n)$ must be adjacent to $c(h_{n-1}) = h_{n-1}$ and $c(w_1^v) = h_{n+1}$, this implies that $c(h_n) = h_n$. Thus we have $c(w_1^v) = h_{n+1}$ and $c(h_n) = h_n$, implying that $\{h_n, h_{n+1}\} \subset c(W)$, which is a contradiction.

Now consider an edge $y_{n-1}^v y_n^v$, with $v \in V(G-H)$. Suppose that $c(\{y_{n-1}^v, y_n^v\}) = \{h_n, h_{n+1}\}$. Without loss of generality, let $c(y_{n-1}^v) = h_{n+1}$ and $c(y_n^v) = h_n$ (due to symmetry). We have $c(u_1^v) = h_1, h_0$, or h_{k-1}. Since $d_{G'}(u_1^v, y_{n-1}^v) = n-1 < d_H(\{h_0, h_1\}, c(y_{n-1}^v) = h_{n+1}) = n$, this implies that $c(u_1^v) \notin \{h_0, h_1\}$, and hence $c(u_1^v) = h_{k-1}$. Since $c(w_1^v)$ must be adjacent to $c(y_n^v) = h_n$, this implies that

$c(w_1^v) \in \{h_{n-1}, h_n, h_{n+1}\}$ (we are not making use of the fact that $c(w_1^v)$ is adjacent to y_{n-1}^v, for $k > 5$, to cover up the case for $k = 5$ also). Since $d_{G'}(u_1^v, w_1^v) = n-1 < d_H(c(u_1^v) = h_{k-1}, \{h_n, h_{n-1}\}) = n$, this implies that $c(w_1^v) \notin \{h_n, h_{n-1}\}$, and hence $c(w_1^v) = h_{n+1}$. Thus we have $c(y_n^v) = h_n$ and $c(w_1^v) = h_{n+1}$, which we have already proved is impossible.

Now consider an edge $vv' \in E(G-H)$. Suppose that $c(\{v, v'\}) = \{h_n, h_{n+1}\}$. Without loss of generality, let $c(v) = h_n$ and $c(v') = h_{n+1}$ (due to symmetry). We have $c(u_1^v), c(u_1^{v'}) \in \{h_1, h_0, h_{k-1}\}$. Since $d_{G'}(u_1^v, v) = n-1 < d_H(\{h_0, h_{k-1}\}, c(v) = h_n) = n$, this implies that $c(u_1^v) \notin \{h_0, h_{k-1}\}$, and hence $c(u_1^v) = h_1$. Since $d_{G'}(u_1^{v'}, v') = n-1 < d_H(\{h_0, h_1\}, c(v') = h_{n+1}) = n$, this implies that $c(u_1^{v'}) \notin \{h_0, h_1\}$, and hence $c(u_1^{v'}) = h_{k-1}$. We have the path $u_1^v u_2^v \dots u_{n-1}^v v$ of length $n-1$ in G'. Since $c(u_1^v) = h_1$ and $c(v) = h_n$, this implies that $c(u_i^v) = h_i$ for all $i = 1, 2, \dots, n-1$. Thus we have $c(u_{n-1}^v) = h_{n-1}$. Since $c(w_{n-1}^v)$ must be adjacent to $c(v) = h_n$, this implies that $c(w_{n-1}^v) \in \{h_{n-1}, h_n, h_{n+1}\}$. Since $d_{G'}(u_1^v, w_{n-1}^v) = n-1 < d_H(c(u_1^v) = h_1, h_{n+1}) = n$, this implies that $c(w_{n-1}^v) \neq h_{n+1}$, and hence $c(w_{n-1}^v) \in \{h_{n-1}, h_n\}$. Since $c(w_{n-1}^{v'})$ must be adjacent to $c(v') = h_{n+1}$, this implies that $c(w_{n-1}^{v'}) \in \{h_n, h_{n+1}, h_{n+2}\}$. Since $d_{G'}(u_1^{v'}, w_{n-1}^{v'}) = n-1 < d_H(c(u_1^{v'}) = h_{k-1}, h_n) = n$, this implies that $c(w_{n-1}^{v'}) \neq h_n$, and hence $c(w_{n-1}^{v'}) \in \{h_{n+1}, h_{n+2}\}$.

We now prove that $c(w_{n-1}^{v'}) \neq h_{n+1}$, which would leave us with the only choice that $c(w_{n-1}^{v'}) = h_{n+2}$. Suppose that $c(w_{n-1}^{v'}) = h_{n+1}$. We have the path $u_1^{v'} u_2^{v'} \dots u_{m+1}^{v'} w_{m+1}^{v'} w_{m+2}^{v'} \dots w_{n-1}^{v'}$ of length $n-1$ in G'. Since $c(u_1^{v'}) = h_{k-1}$ and $c(w_{n-1}^{v'}) = h_{n+1}$, this implies that $c(u_i^{v'}) = h_{k-i}$ for all $i = 1, 2, \dots, m+1$, and $c(w_i^{v'}) = h_{k-i-1}$ for all $i = m+1, m+2, \dots, n-1$. Thus we have $c(u_m^{v'}) = h_{k-m}$ and $c(w_{m+1}^{v'}) = h_{k-m-2}$. Since $w_m^{v'}$ is adjacent to $w_{m+1}^{v'}$ and at distance two from $u_m^{v'}$ in G', this implies that $c(w_m^{v'}) = h_{k-m-2}$ or h_{k-m-1}. Since $w_1^{v'}$ is at distance $m-1$ from $w_m^{v'}$ in G', it follows that $c(w_1^{v'}) = h_s$, $n+1 \leq s \leq k-2$. We also have the path $u_1^v u_2^v \dots u_{m+1}^v w_{m+1}^v w_{m+2}^v \dots w_{n-1}^v$ of length $n-1$ in G'. As shown above, we have $c(u_i^v) = h_i$, for all $i = 1, 2, \dots, m+1$, and $c(w_{n-1}^v) = h_n$ or h_{n-1}. This implies that $c(w_i^v) = h_{i+1}$ or h_i for all $i = m+1, m+2, \dots, n-1$. Thus we have $c(u_m^v) = h_m$ and $c(w_{m+1}^v) = h_{m+1}$ or h_{m+2}. Since w_m^v is adjacent to w_{m+1}^v and at distance two from u_m^v in G', this implies that $c(w_m^v) = h_m, h_{m+1}$, or h_{m+2}. Since w_1^v is at distance $m-1$ from w_m^v in G', it follows that $c(w_1^v) = h_t$, $1 \leq t \leq n$. We have that $c(w_1^v)$ and $c(w_1^{v'})$ must be adjacent in H. The only possible pair of values for t and s are n and $n+1$, respectively, for $c(w_1^v)$ to be adjacent to $c(w_1^{v'})$. Thus $c(w_1^v) = h_n$ and $c(w_1^{v'}) = h_{n+1}$, which implies that $h_n, h_{n+1} \in c(W)$ and we have a contradiction. Thus $c(w_{n-1}^{v'}) \neq h_{n+1}$, and hence $c(w_{n-1}^{v'}) = h_{n+2}$.

Now, $c(x_{vv'})$ must be adjacent to $c(u_{n-1}^v) = h_{n-1}$, $c(v) = h_n$, $c(v') = h_{n+1}$, and $c(w_{n-1}^{v'}) = h_{n+2}$, which is impossible.

For the edges $v'x_{vv'}$, $w_{n-1}^{v'}x_{vv'}$, with $vv' \in E(G-H)$; and a possible edge vh_n, with $v \in V(G-H)$, the arguments are exactly as in the case $k = 4m+2$.

Finally, consider a possible edge vh_{n+1}, with $v \in V(G-H)$. Suppose that $c(\{v, h_{n+1}\}) = \{h_n, h_{n+1}\}$. Without loss of generality, let $c(v) = h_n$ and $c(h_{n+1}) = h_{n+1}$ (due to symmetry). Exactly, as in the case $vv' \in E(G-H)$ above, we establish that $c(w_1^v) = h_t$, $1 \leq t \leq n$. Since w_1^v is at distance two from h_{n+1} in G', it follows that $c(w_1^v) = h_n$ or h_{n-1}. Clearly, $c(w_1^v) \neq h_n$, as otherwise we will have $h_n, h_{n+1} \in c(W)$. Thus $c(w_1^v) = h_{n-1}$. Since h_n is adjacent to w_1^v and h_{n+1}, this implies that $c(h_n) = h_n$, and hence $h_n, h_{n+1} \in c(W)$, which is a contradiction.

This completes the proof that $\{h_n, h_{n+1}\} \subset c(W)$. Thus $c(W) = \{h_{n-1}, h_n, h_{n+1}\}$

or $c(W) = \{h_n, h_{n+1}, h_{n+2}\}$. Without loss of generality, suppose that $c(W) = \{h_{n-1}, h_n, h_{n+1}\}$ (due to symmetry). This implies that $c(h_n) = h_n$, as h_n is adjacent to every vertex in W. We have $c(U) = \{h_1, h_0, h_{k-1}\}$ and h_0 is adjacent to every vertex in U. Hence $c(h_0) = h_0$. We have the path $h_0h_1 \dots h_n$ of length n in G'. Since $c(h_0) = h_0$ and $c(h_n) = h_n$, this implies that $c(h_i) = h_i$ for all $i = 0, 1, \dots, n$.

We now prove that $c(h_{k-1}) = h_{k-1}$. Suppose to the contrary that $c(h_{k-1}) \neq h_{k-1}$. Thus we have $c(h_{k-1}) = h_0$ or h_1, $c(h_0) = h_0$, and $c(h_1) = h_1$. Since $c(U) = \{h_1, h_0, h_{k-1}\}$, it must be that $c(u_1^v) = h_{k-1}$ for some vertex v of $G - H$. We have $c(w_1^v) \in \{h_{n-1}, h_n, h_{n+1}\}$, as $c(W) = \{h_{n-1}, h_n, h_{n+1}\}$. Since $d_{G'}(u_1^v, w_1^v) = n - 1 < d_H(c(u_1^v) = h_{k-1}, \{h_n, h_{n-1}\}) = n$, this implies that $c(w_1^v) \notin \{h_n, h_{n-1}\}$, and hence $c(w_1^v) = h_{n+1}$. Now, $c(y_n^v)$ must be adjacent to $c(h_{n-1}) = h_{n-1}$ and $c(w_1^v) = h_{n+1}$. Hence $c(y_n^v) = h_n$. Since $c(y_1^v)$ must be adjacent to $c(u_1^v) = h_{k-1}$ and $c(h_{k-1}) = h_0$ or h_1, this implies that $c(y_1^v) \in \{h_0, h_{k-1}\}$. However, $d_{G'}(y_1^v, y_n^v) = n - 1 < d_H(\{h_0, h_{k-1}\}, c(y_n^v) = h_n) = n$, and hence it is impossible that $c(y_1^v) \in \{h_0, h_{k-1}\}$. Hence it must be that $c(h_{k-1}) = h_{k-1}$.

We have the path $h_{k-1}h_{k-2} \dots h_n$ of length n in G'. Since $c(h_{k-1}) = h_{k-1}$ and $c(h_n) = h_n$, this implies that $c(h_i) = h_i$ for all $i = n, n+1, \dots, k-1$. Thus we have $c(h_i) = h_i$ for all $i = 0, 1, 2, \dots, k-1$. Hence $c : G' \to H$ is a retraction, and the lemma is proved. □

Proof of Theorem 3.1 in the case $k = 4m + 1$. The construction of G' is as follows. For each vertex v in $G - H$, we add to G three vertex disjoint paths U_v, W_v, and Y_v, where U_v and W_v each contains $n - 1$ new vertices, and Y_v contains n new vertices. Let $U_v = u_1^v u_2^v \dots u_{n-1}^v$, $W_v = w_1^v w_2^v \dots w_{n-1}^v$, and $Y_v = y_1^v y_2^v \dots y_n^v$, with $v \in V(G - H)$. We add the edges $h_0u_1^v$, vu_{n-1}^v, $h_nw_1^v$, vw_{n-1}^v, $u_m^v w_m^v$, $h_{k-1}y_1^v$, $h_{n-1}y_n^v$, $u_1^v y_1^v$, and $w_1^v y_n^v$, with $v \in V(G - H)$. Further, if $k > 5$ then we also add the edge $w_1^v y_{n-1}^v$, with $v \in V(G - H)$. If $k = 5$ then for each $v \in V(G - H)$ we add a new vertex z_v adjacent to y_{n-1}^v, h_n, and u_1^v (note the similarity between the adjacencies of z_v and w_1^v). Thus there is a slight difference in the construction of G' for $k = 4m + 1 > 5$ and $k = 4m + 1 = 5$. See Figures 3.6 and 3.7.

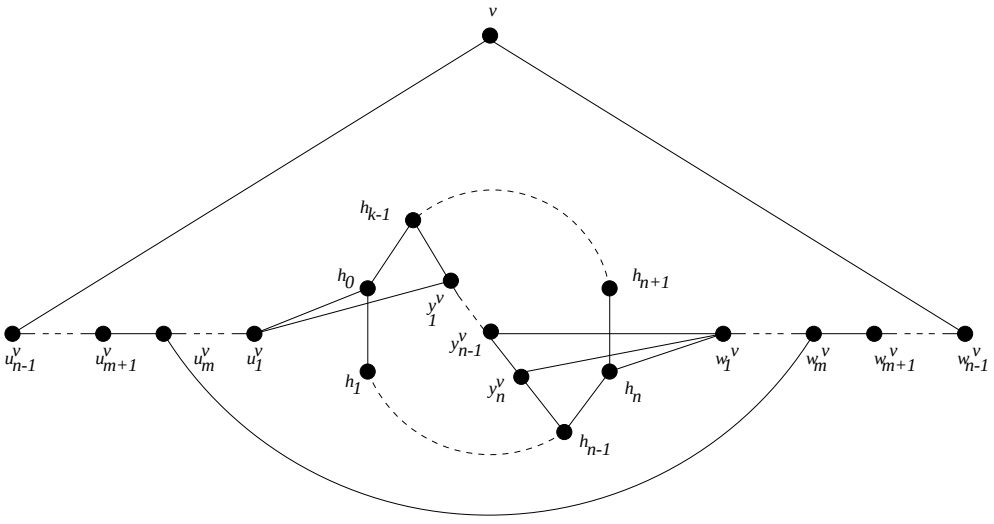


FIG. 3.6. *Construction of G' for a vertex v in $G - H$, with $k = 4m + 1 > 5$.*

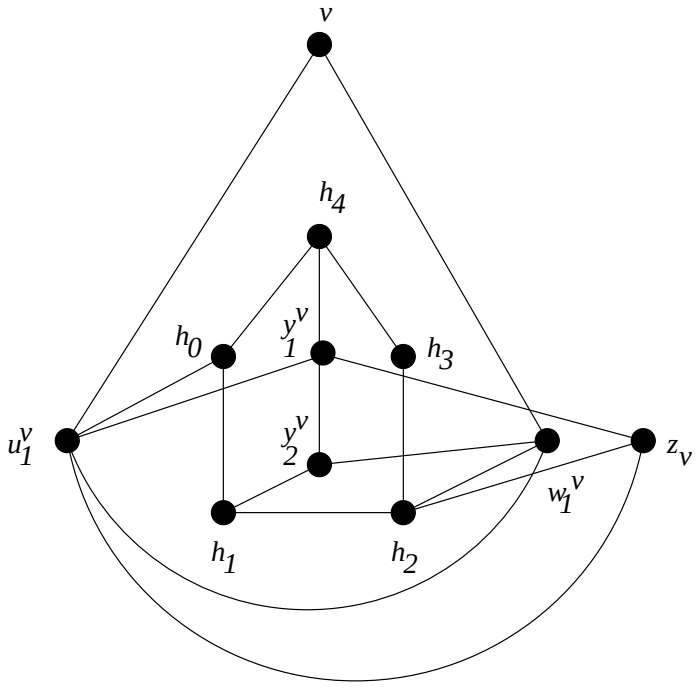


FIG. 3.7. *Construction of G' for a vertex v in $G - H$, with $k = 4m + 1 = 5$.*

The addition of a new vertex $x_{vv'}$ and the edges associated with it are same as for the case $k = 4m+2$, with $vv' \in E(G-H)$, $v \neq v'$. See Figure 3.3. This completes the construction of G'.

Note the similarity between the construction of G' for $k = 4m+1$ and $k = 4m+3$. The two lemmas that we gave for the case when $k = 4m + 3$ hold similarly for the case when $k = 4m + 1$ with some variations in their proofs as noted below.

LEMMA 3.1.7. *G retracts to H if and only if G' retracts to H.*

Proof. If G' retracts to H then clearly G retracts to H, as G is a subgraph of G'. If $r : G \to H$ is a retraction then we define a retraction $r' : G' \to H$ exactly as we defined in the case $k = 4m + 3$ in Lemma 3.1.5 except that now we do not need to define $r'(a_v)$, with $v \in V(G - H)$. For $k = 5$ however, we now need to define $r'(z_v)$, with $v \in V(G - H)$: in the case $0 \leq j \leq n$, we define $r'(z_v) = h_{n-1} = h_1$; and in the case $n + 1 \leq j \leq k - 1$, we define $r'(z_v) = h_{n+1} = h_3$. As for the case $k = 4m + 2$, it can be verified that for every edge ab of G', $r'(a)r'(b)$ is indeed an edge of H. Recall that the edges of G' are (note the difference in the set of edges as compared to the case $k = 4m$) ab, $u_i^v u_{i+1}^v$, $w_i^v w_{i+1}^v$, $y_j^v y_{j+1}^v$, $u_1^v h_0$, $w_1^v h_n$, $v u_{n-1}^v$, $v w_{n-1}^v$, $u_m^v w_m^v$, $y_1^v h_{k-1}$, $y_n^v h_{n-1}$, $u_1^v y_1^v$, $w_1^v y_n^v$, $g x_{gg'}$, $g' x_{gg'}$, $x_{gg'} u_{n-1}^g$, and $x_{gg'} w_{n-1}^{g'}$, with $ab \in E(G)$, $v \in V(G - H)$, $gg' \in E(G - H)$, $i = 1, 2, \ldots, n - 2$, $j = 1, 2, \ldots, n - 1$. Further, if $k > 5$ then G' also has the edge $w_1^v y_{n-1}^v$, with $v \in V(G - H)$. If $k = 5$ then G' also has the edges $z_v y_{n-1}^v$, $z_v h_n$, and $z_v u_1^v$, with $v \in V(G - H)$. □

LEMMA 3.1.8. *G' retracts to H if and only if G' compacts to H.*

Proof. If G' retracts to H then by definition G' compacts to H. Now suppose that $c : G' \to H$ is a compaction. We shall prove that G' retracts to H. For $k > 5$, we define the sets U and W as we did for the case $k = 4m + 3$ in Lemma 3.1.6, i.e., we let $U =$

$\{u_1^v | v \in V(G-H)\} \cup \{h_1, h_0, h_{k-1}\}$ and $W = \{w_1^v | v \in V(G-H)\} \cup \{h_{n-1}, h_n, h_{n+1}\}$. For $k = 5$, the set U is same as defined here but the set W now also contains z_v for all $v \in V(G-H)$, i.e., $W = \{w_1^v, z_v | v \in V(G-H)\} \cup \{h_{n-1}, h_n, h_{n+1}\}$.

We note that $d_{G'}(U, a) < n$ and $d_{G'}(W, a) < n$ for all $a \in V(G')$. We point out here that an upper bound on $d_{G'}(W, y)$, for the vertices y of Y_v, with $v \in V(G-H)$, is obtained differently in the cases $k > 5$ and $k = 5$ due to the following paths in G':
for $k > 5$: $w_1^v y_{n-1}^v y_{n-2}^v \dots y_1^v$, and $w_1^v y_n^v$;
for $k = 5$: $z_v y_{n-1}^v y_{n-2}^v \dots y_1^v = z_v y_1^v$, and $w_1^v y_n^v = w_1^v y_2^v$.

Thus, exactly as for the case $k = 4m+3$, we establish that both $c(U)$ and $c(W)$ induce paths containing three vertices in H. Without loss of generality, suppose that $c(U) = \{h_1, h_0, h_{k-1}\}$ (due to symmetry). We first prove that $\{h_n, h_{n+1}\} \subset c(W)$, i.e., $c(W) = \{h_{n-1}, h_n, h_{n+1}\}$ or $c(W) = \{h_n, h_{n+1}, h_{n+2}\}$. Let some edge ab of G' cover the edge $h_n h_{n+1}$ of H under c. As for the case $k = 4m+3$, we have that both a and b are at distance $n-1$ from U in G'. The only vertices that could possibly be at distance $n-1$ from U in G' are h_n, h_{n+1}, w_1^v, w_{n-1}^v, y_{n-1}^v, y_n^v, v, with $v \in V(G-H)$; $x_{vv'}$, with $vv' \in E(G-H)$; and for $k = 5$, we also have z_v, with $v \in V(G-H)$. Upper bounds on distance to these vertices from U in G' may be obtained due to the following paths in G':
$h_1 h_2 \dots h_n$, $h_{k-1} h_{k-2} \dots h_{n+1}$, $u_1^v u_2^v \dots u_m^v w_m^v w_{m-1}^v \dots w_1^v$,
$u_1^v u_2^v \dots u_m^v w_m^v w_{m+1}^v \dots w_{n-1}^v$, $u_1^v y_1^v y_2^v \dots y_{n-1}^v$, $h_1 h_2 \dots h_{n-1} y_n^v$,
$u_1^v u_2^v \dots u_{n-1}^v v$, and $u_1^v u_2^v \dots u_{n-1}^v x_{vv'}$.
Notice that the paths to w_1^v and w_{n-1}^v from u_1^v, mentioned above, are different from the case $k = 4m+3$.

Suppose that $\{h_n, h_{n+1}\} \not\subset c(W)$. Then, just as for the case $k = 4m+3$, we conclude that ab must be an edge among $v w_{n-1}^v$, $w_1^v y_n^v$, $y_{n-1}^v y_n^v$, with $v \in V(G-H)$; vv', $v x_{vv'}$, $v' x_{vv'}$, $w_{n-1}^{v'} x_{vv'}$, with $vv' \in E(G-H)$; $w_1^v y_{n-1}^v$, when $k > 5$, with $v \in V(G-H)$; and $z_v y_{n-1}^v$, when $k = 5$, with $v \in V(G-H)$; in order to meet the requirements of the edge ab, both vertices in each of these edges are assumed to achieve distance $n-1$ from U in G'. Further, if $v h_n$ or $v h_{n+1}$ is an edge of G, for some vertex $v \in V(G-H)$, then we need to include such an edge also for ab. Note that the set of these possible edges for ab is the same as for the case $k = 4m+3$ except for a difference in the case $k = 5$. For all these possible edges of ab except $vv' \in E(G-H)$ and $z_v y_{n-1}^v$, when $k = 5$, with $v \in V(G-H)$, the proof that these edges do not cover the edge $h_n h_{n+1}$ under c (i.e., $c(\{a, b\}) \neq \{h_n, h_{n+1}\}$) is exactly the same as for the case $k = 4m+3$.

For an edge $z_v y_{n-1}^v$, when $k = 5$, we argue similarly as for an edge $v w_{n-1}^v$, with $v \in V(G-H)$, to show that it does not cover the edge $h_n h_{n+1}$ under c.

For an edge vv', we now make use of the paths $u_1^v u_2^v \dots u_m^v w_m^v w_{m+1}^v \dots w_{n-1}^v$ and $u_1^{v'} u_2^{v'} \dots u_m^{v'} w_m^{v'} w_{m+1}^{v'} \dots w_{n-1}^{v'}$ instead of the paths $u_1^v u_2^v \dots u_{m+1}^v w_{m+1}^v w_{m+2}^v \dots w_{n-1}^v$ and $u_1^{v'} u_2^{v'} \dots u_{m+1}^{v'} w_{m+1}^{v'} w_{m+2}^{v'} \dots w_{n-1}^{v'}$, respectively, with $vv' \in E(G-H)$. We consider this case and show its proof.

Let vv' be an edge of $G-H$. Suppose that $c(\{v, v'\}) = \{h_n, h_{n+1}\}$. Without loss of generality, let $c(v) = h_n$ and $c(v') = h_{n+1}$ (due to symmetry). Exactly, as in the case $k = 4m+3$, we prove that $c(u_i^v) = h_i$ for all $i = 1, 2, \dots, n-1$, $c(u_1^{v'}) = h_{k-1}$, $c(w_{n-1}^v) \in \{h_{n-1}, h_n\}$, and $c(w_{n-1}^{v'}) \in \{h_{n+1}, h_{n+2}\}$.

Now we prove that $c(w_{n-1}^{v'}) \neq h_{n+1}$, which would leave us with the only choice that $c(w_{n-1}^{v'}) = h_{n+2}$. Suppose that $c(w_{n-1}^{v'}) = h_{n+1}$. We have the path $u_1^{v'} u_2^{v'} \dots u_m^{v'} w_m^{v'} w_{m+1}^{v'} \dots w_{n-1}^{v'}$ of length $n-1$ in G'. Since $c(u_1^{v'}) = h_{k-1}$ and $c(w_{n-1}^{v'})$

$= h_{n+1}$, this implies that $c(u_i^{v'}) = h_{k-i}$ for all $i = 1, 2, \ldots, m$, and $c(w_i^{v'}) = h_{k-i-1}$ for all $i = m, m+1, \ldots, n-1$. Thus we have $c(w_m^{v'}) = h_{k-m-1}$. Since $w_1^{v'}$ is at distance $m-1$ from $w_m^{v'}$ in G', it follows that $c(w_1^{v'}) = h_s$, $n+1 \leq s \leq k-2$. We also have the path $u_1^v u_2^v \ldots u_m^v w_m^v w_{m+1}^v \ldots w_{n-1}^v$ of length $n-1$ in G'. Since $c(u_i^v) = h_i$ for all $i = 1, 2, \ldots, m+1$, and $c(w_{n-1}^v) = h_n$ or h_{n-1}, this implies that $c(w_i^v) = h_{i+1}$ or h_i, for all $i = m, m+1, \ldots, n-1$. Thus we have $c(w_m^v) = h_{m+1}$ or h_m. Since w_1^v is at distance $m-1$ from w_m^v in G', it follows that $c(w_1^v) = h_t$, $1 \leq t \leq n$. We have that $c(w_1^v)$ and $c(w_1^{v'})$ must be adjacent in H. The only possible pair of values for t and s are n and $n+1$, respectively, for $c(w_1^v)$ to be adjacent to $c(w_1^{v'})$. Thus $c(w_1^v) = h_n$ and $c(w_1^{v'}) = h_{n+1}$, which implies that $h_n, h_{n+1} \in c(W)$ and we have a contradiction. Thus $c(w_{n-1}^{v'}) \neq h_{n+1}$, and hence $c(w_{n-1}^{v'}) = h_{n+2}$.

Now, $c(x_{vv'})$ must be adjacent to $c(u_{n-1}^v) = h_{n-1}$, $c(v) = h_n$, $c(v') = h_{n+1}$, and $c(w_{n-1}^{v'}) = h_{n+2}$, which is impossible. Hence $c(\{v, v'\}) \neq \{h_n, h_{n+1}\}$.

Thus we have that $\{h_n, h_{n+1}\} \subset c(W)$. Hence $c(W) = \{h_{n-1}, h_n, h_{n+1}\}$ or $c(W) = \{h_n, h_{n+1}, h_{n+2}\}$. Without loss of generality, suppose that $c(W) = \{h_{n-1}, h_n, h_{n+1}\}$ (due to symmetry). Exactly, as for the case $k = 4m+3$, we prove that $c(h_{k-1}) = h_{k-1}$ and establish that $c(h_i) = h_i$ for all $i = 0, 1, 2, \ldots, k-1$, i.e., $c : G' \to H$ is a retraction. □

We have thus proved Theorem 3.1. □

4. Compaction to an irreflexive *k*-cycle. In this section, we shall assume that C_k denotes an irreflexive k-cycle.

It follows from [Hell and Nesetril, 1990] that *RET-*C_k and *COMP-*C_k are both NP-complete for all odd $k \geq 3$. It is easy to see that *RET-*C_4, and hence *COMP-*C_4, are both polynomial time solvable. In fact, when H is a chordal bipartite graph (which includes C_4), the problem *RET-H* is polynomial time solvable [Bandelt, Dahlmann, and Schutte, 1987], and hence *COMP-H* is also polynomial time solvable. Feder showed that *RET-*C_k is NP-complete for all even $k \geq 6$ (personal communication through Hell, cf. [Feder, Hell, and Huang, 1999]). This was also proved independently by G. MacGillivray in 1988 (personal communication). The transformation used by both is from the $k/2$-colorability problem.

Determining the complexity of *COMP-*C_k, for any particular even $k \geq 6$, has been of interest since a long time to various people including Hell and Nesetril (personal communications). We show that *COMP-*C_k is NP-complete for all even $k \geq 6$. To show this, we give a transformation from *RET-*C_k to *COMP-*C_k, using the technique described in section 1 where we restrict the input graph to be bipartite for all even $k \geq 6$. For proving NP-completeness of *COMP-*C_k, for all even $k \geq 6$, we provide two categories of construction depending on whether $k = 4m+2$ for some integer $m \geq 1$, or $k = 4m$ for some integer $m \geq 2$.

Acknowledgments. The author would like to thank Professor Dr. Pavol Hell for his suggestions on improving the style of presentation for the work.

The author obtained the results for the open problem as a result of a direct encouragement from his father Professor Dr. Sukhdeo Narayan Prasad. The author would like to thank his father for his thoughtful encouragement with a remarkable effect.

REFERENCES

H. J. Bandelt, A. Dahlmann, and H. Schutte (1987), *Absolute retracts of bipartite graphs*, Discrete Appl. Math., 16, pp. 191–215.

H. J. Bandelt, M. Farber, and P. Hell (1993), *Absolute reflexive retracts and absolute bipartite retracts*, Discrete Appl. Math., 44, pp. 9–20.
T. Feder and P. Hell (1998), *List homomorphisms to reflexive graphs*, J. Combin. Theory Ser. B, 72, pp. 236–250.
T. Feder, P. Hell, and J. Huang (1999), *List homomorphisms and circular arc graphs*, Combinatorica, 19, pp. 487–505.
T. Feder and P. Winkler (1988), manuscript.
F. Harary (1969), *Graph Theory*, Addison-Wesley, Reading, MA.
P. Hell (1972), *Retractions de Graphes*, Ph.D. thesis, Universite de Montreal, Montreal, Quebec, Canada.
P. Hell (1974), *Retracts in graphs*, in Graphs and Combinatorics, Lecture Notes in Math. 406, Springer-Verlag, Berlin, pp. 291–301.
P. Hell and D. J. Miller (1979), *Graphs with forbidden homomorphic images*, Ann. New York Acad. Sci., 319, pp. 270–280.
P. Hell and J. Nesetril (1990), *On the complexity of H-coloring*, J. Combin. Theory Ser. B, 48, pp. 92–110.
P. Hell and I. Rival (1987), *Absolute retracts and varieties of reflexive graphs*, Canad. J. Math., 39, pp. 544–567.
A. Karabeg and D. Karabeg (1991), *Graph Compaction*, in Graph Theory Notes of New York XXI, The New York Academy of Sciences, pp. 44–51.
A. Karabeg and D. Karabeg (1993), *Graph Compaction*, manuscript.
R. Nowakowski and I. Rival (1979), *Fixed-edge theorem for graphs with loops*, J. Graph Theory, 3, pp. 339–350.
E. Pesch (1988), *Retracts of Graphs*, Math. Systems Econom. 110, Athenaum, Frankfurt am Main, Germany.
E. Pesch and W. Poguntke (1985), *A characterization of absolute retracts of n-chromatic graphs*, Discrete Math., 57, pp. 99–104.
N. Vikas (1999), *Computational complexity of compaction to cycles*, in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 281–306

BACKWARD CONSISTENCY AND SENSE OF DIRECTION IN ADVANCED DISTRIBUTED SYSTEMS*

P. FLOCCHINI†, A. RONCATO‡, AND N. SANTORO§

Abstract. Studies on the relationship between label consistency, computability, and complexity assume the existence of *local orientation*; this assumption is in fact at the basis of the *point-to-point* model and is realistic for systems where a communication link can connect only two entities. However, in systems which use more advanced communication and interconnection technology, such as *buses*, *optical* networks, and *wireless* communication media, and more importantly, in heterogeneous systems (such as the *Internet*) which include any combination of the above, local orientation *cannot* be assumed. This implies that the entire established body of results on the relationship between label consistency (e.g., *sense of direction*) and computability and complexity does not hold for systems with advanced communication technology.

In this paper we consider a new type of consistency which we shall call *backward consistency* and which, unlike sense of direction, can exist even without local orientation. Thus, unlike all previous forms of consistency, it can be found (or designed) in advanced distributed systems.

We study backward consistency both in terms of its relationship with the traditional properties of local orientation and (weak) sense of direction, and with respect to symmetries of the edge labelings and of the naming functions. We show that backward consistency is computationally equivalent to sense of direction; in other words, it is possible to take advantage of the computational power of sense of direction even in the absence of local orientation.

Key words. distributed computing, sense of direction, global consistency

AMS subject classifications. 05C78, 68Q10, 68R10, 68W15

PII. S0097539700377293

1. Introduction.

A distributed system is a collection of computational entities communicating by exchanging messages. Depending on how the communication is achieved, different models exist. In the *point-to-point* model, the communication topology of the system is viewed as an edge-labeled undirected graph $(G = (V, E), \lambda)$ where nodes correspond to the system entities, edges represent pairs of neighboring entities (i.e., entities which can communicate directly), each node $x \in V$ has a local label (usually called port number) $\lambda_x(\langle x, y\rangle)$ associated to each of its incident edges $\langle x, y\rangle$, and $\lambda = \{\lambda_x : x \in V\}$ is the set of labeling functions.

In studies on computing in distributed systems, recurring challenging questions arise concerning the relationships between *local views* and *global consistency*, and concerning their impact on distributed computability and communication complexity. In these studies, *locality* is at the entity level: the local view of an entity x consists of just the labeling λ_x of its communication ports (or communication links, incident edges) and, sometimes, of the naming β_x used to refer to the other entities. A major goal of this investigation is to derive under what properties of λ and/or β it is possible to

*Received by the editors August 22, 2000; accepted for publication (in revised form) August 5, 2002; published electronically January 17, 2003. This research was supported in part by NSERC. A preliminary version of this paper appeared in *Proceedings of the* 18*th ACM Symposium on Principles of Distributed Computing* (PODC 99).

http://www.siam.org/journals/sicomp/32-2/37729.html

†School of Information Technology and Engineering, University of Ottawa, 800 King Eduard, Ottawa K1N 6N5, Canada (flocchin@site.uottawa.ca).

‡Dipartimento di Informatica, Università di Venezia, Via Torino 155, 30172 Mestre, Italy (roncato @dsi.unive.it).

§School of Computer Science, Carleton University, Ottawa K1S 5B6, Canada (santoro@scs. carleton.ca).

infer global properties about the entire system. A practical aim is to determine if and how these properties can be usefully employed, e.g., to yield more efficient distributed computations. Examples of these studies include *compact routing,* which focuses on determining under what conditions the local port numbering provides global routing information for given global naming schemes (see [25] for a recent survey), and *sense of direction,* which analyzes the impact on communication complexity of local port labelings and local naming schemes (see [21] for a recent survey).

In particular, there is a large body of evidence on the positive impact on complexity of the set of global consistency constraints satisfied by labelings with sense of direction (e.g., see [18, 19, 22, 30, 32, 33, 41]). Properties of sense of direction have been studied, for example, in [5, 8, 10, 11, 17, 20, 23, 24, 26, 42]. Several levels of sense of direction have been considered and investigated, with researchers trying to identify the "smallest" amount of consistency really needed (e.g., to efficiently solve a specific problem).

All studies in the relevant literature, with very few exceptions, make the basic (but often unstated) fundamental assumption that the entities are able to distinguish among their incident links, i.e., the labelings λ_x are injective functions. Even if the links haven't yet been assigned a label, it is assumed that an entity is capable of such an assignment (e.g., [15, 27, 31]). This assumption, usually called *local orientation*, is in fact at the basis of the *point-to-point* model and is realistic for systems where a communication link can connect only two entities.

If we are to model systems which use more advanced communication and interconnection technology, such as *buses*, *optical* networks, and *wireless* communication media, and more importantly, in heterogeneous systems (such as the *Internet*) which include any combination of the above, local orientation *cannot* be assumed. This is because any direct connection between k entities will correspond, at each of those entities, to $k-1$ edges with the same label; hence, if $k > 2$ (e.g., a bus), λ is not injective. That is, unless the system is point-to-point (i.e., each connection is between only two entities) there are nodes that cannot distinguish between some of their incident edges. This implies that the entire established body of results on the relationship between consistency, computability, and complexity does not hold and, thus, cannot be applied outside the point-to-point model. In other words, very little is known on this subject for systems with advanced communication technology. (A vast body of knowledge does indeed exist on these systems, but not on these topics.)

There are a few exceptions: the early work on computing in complete graphs with "ambiguous" labelings [26, 40], the more recent studies on computability in anonymous "wireless" networks [14, 37], and the results on leader election and function evaluation in anonymous networks with different types of "port awareness" [9, 45]. Note that "ambiguous," "wireless," and "port aware" all denote the (possible) absence of local orientation.

In this paper we consider a new type of consistency which we shall call *backward consistency*. Backward consistency is strongly related to the classical "forward" consistency implied by sense of direction; however, unlike sense of direction, it can exist even without local orientation. Thus, unlike all previous forms of consistency, backward consistency can be found (or designed) in advanced distributed systems.

In the following, we study backward consistency both in terms of its relationship with the traditional properties of local orientation and (weak) sense of direction, and with respect to symmetries of the edge labelings and of the naming functions. In particular, we focus on the conditions under which a labeling can have simultaneously forward and backward consistency.

We then consider the power of backward consistency and prove that *backward consistency is computationally equivalent to sense of direction.* In other words, it is possible to take advantage of the computational power of sense of direction even in the absence of local orientation. This counterintuitive result somehow extends and generalizes the results of [1, 14] on the insensitivity of ring networks to local orientation. In addition to the theoretical result, we also provide a simple effective procedure to efficiently map protocols exploiting sense of direction into ones using "backward sense of direction."

In the following we consider the undirected case (i.e., bidirectional communication capabilities); this is done only for simplicity of exposition, as all results extend to and hold in the directed case as well.

2. Basic definitions and properties.

2.1. Labelings and sense of direction. In this section we first recall the basic definitions of consistency, symmetry, and sense of direction in labeled graphs; most of the terminology follows [20].

Let $G = (V, E)$ be a simple connected undirected graph, and let $E(x)$ denote the set of edges incident to node $x \in V$. Given $G = (V, E)$ and a set Σ of labels, a *local labeling function* of $x \in V$ is any function $\lambda_x : E(x) \to \Sigma$ which associates a label $l \in \Sigma$ to each edge $e \in E(x)$. A set $\lambda = \{\lambda_x : x \in V\}$ of local labeling functions will be called a *labeling* of G, and by (G, λ) we shall denote the corresponding *(edge-)labeled graph.* If all functions in λ are injective, we say that λ is a *local orientation.* Note that local orientation is assumed in the point-to-point model and in almost all literature on informative labelings, including those on unlabeled networks (e.g., [15, 27, 31]). A labeling λ is *symmetric* if there exists a bijection $\psi : \Sigma \to \Sigma$ such that for each $\langle x, y\rangle \in E$, $\lambda_y(\langle y, x\rangle) = \psi(\lambda_x(\langle x, y\rangle))$; ψ will be called the edge-symmetry function.

A *walk* π in G is a sequence of edges in which the endpoint of one edge is the starting point of the next edge. Let $P[x]$ denote the set of all walks starting from $x \in V$, and let $P[x, y]$ denote the set of walks starting from $x \in V$ and ending in $y \in V$. Let $\Lambda_x : P[x] \to \Sigma^+$ and $\Lambda = \{\Lambda_x : x \in V\}$ denote the extension of λ_x and λ, respectively, from edges to walks; let $\Lambda[x] = \{\Lambda_x(\pi) : \pi \in P[x]\}$, and let $\Lambda[x, y] = \{\Lambda_x(\pi) : \pi \in P[x, y]\}$.

Given an edge-symmetry function ψ, we shall denote by $\Psi : \Sigma^+ \to \Sigma^+$ its extension to strings; i.e., for $\alpha = a_1 \cdot a_2 \cdot \ldots \cdot a_p \in \Sigma^+$, $\Psi(\alpha) = \psi(a_p) \cdot \ldots \cdot \psi(a_1)$, where $\cdot$ denotes the concatenation operator.

Informally, a labeled graph (G, λ) has *sense of direction* when it is possible to understand, from the labels associated to the edges, whether different walks from any given node x end in the same node or in different ones.

A *coding* function of (G, λ) is any function $\mathbf{c}$ with domain Σ^+. It is said to be *consistent* if the following holds $\forall x, y, z \in V$, $\forall \pi_1 \in P[x, y]$, $\pi_2 \in P[x, z]$: $\mathbf{c}(\Lambda_x(\pi_1)) = \mathbf{c}(\Lambda_x(\pi_2))$ iff $y = z$. In other words, the labelings of the walks *originating* from the same node are mapped to the same value iff they end in the same node. Thus, we can define a consistent coding if the following holds $\forall x, y, z \in V$, $\forall \alpha \in \Lambda[x, y]$, $\beta \in \Lambda[x, z]$: $\mathbf{c}(\alpha) = \mathbf{c}(\beta)$ iff $y = z$. We shall denote by $\mathcal{N}(\mathbf{c})$ the codomain of $\mathbf{c}$.

DEFINITION 1 (weak sense of direction). *A labeled graph (G, λ) has* weak sense of direction $\mathbf{c}$ *iff* $\mathbf{c}$ *is a consistent coding function of (G, λ). Alternatively, we shall say that* $\mathbf{c}$ *is a weak sense of direction in (G, λ).*

We see immediately that consistency requires local orientation, as follows.

LEMMA 1. *Let (G, λ) have weak sense of direction* $\mathbf{c}$*. Then λ has local orientation.*

Proof. By contradiction, let there exist two distinct neighbors y, z of a node $x \in V$ such that $\lambda_x(\langle x, y\rangle) = \lambda_x(\langle x, z\rangle)$; thus, $\mathbf{c}(\lambda_x(\langle x, y\rangle)) = \mathbf{c}(\lambda_x(\langle x, z\rangle))$. On the other hand, by definition of the consistent coding function, $\mathbf{c}(\lambda_x(\langle x, y\rangle)) \neq \mathbf{c}(\lambda_x(\langle x, z\rangle))$, which leads to the contradiction. □

Given a coding function $\mathbf{c}$, a *decoding function* $\mathbf{d}$ for $\mathbf{c}$ is any function $\mathbf{d} : \Sigma \times \mathcal{N}(\mathbf{c}) \to \mathcal{N}(\mathbf{c})$ such that $\forall \langle x, y\rangle \in E(x)$, $\pi \in P[y, z]$: $\mathbf{d}(\lambda_x(\langle x, y\rangle), \mathbf{c}(\Lambda_y(\pi))) = \mathbf{c}(\lambda_x(\langle x, y\rangle) \cdot \Lambda_y(\pi))$, where $\cdot$ is the concatenation operator. Or alternatively, $\forall a \cdot w \in \Lambda[x, z]$, $a \in \Sigma$: $\mathbf{d}(a, \mathbf{c}(w)) = \mathbf{c}(a \cdot w)$.

DEFINITION 2 (sense of direction). *A labeled graph* (G, λ) *has* sense of direction $(\mathbf{c}, \mathbf{d})$ *iff* $\mathbf{c}$ *is a weak sense of direction and* $\mathbf{d}$ *is a decoding function for* $\mathbf{c}$. *Alternatively, we shall say that* $(\mathbf{c}, \mathbf{d})$ *is a sense of direction in* (G, λ).

Let $\mathcal{L}$, $\mathcal{W}$, and $\mathcal{D}$ denote the set of labeled graphs (G, λ) with local orientation, weak sense of direction, and sense of direction, respectively. The following relationship holds.

LEMMA 2 (see [11, 20]). $\mathcal{D} \subset \mathcal{W} \subset \mathcal{L}$.

The proof follows from the fact that local orientation is necessary but (obviously) not sufficient for consistency [20], and the fact that there exist labeled graphs with weak sense of direction but without sense of direction [11].

2.2. Backward consistency and sense of direction. In this section we introduce the notion of backward consistency.

DEFINITION 3. *A coding function* $\mathbf{c}$ *of* (G, λ) *is* backward consistent *if the following holds* $\forall x, y, z \in V$, $\forall \pi_1 \in P[x, z]$, $\pi_2 \in P[y, z]$: $\mathbf{c}(\Lambda_x(\pi_1)) = \mathbf{c}(\Lambda_y(\pi_2)) \Leftrightarrow x = y$.

In other words, sequences of labels of walks *terminating* in the same node are mapped to the same value iff they start from the same node.

Note that the difference between the consistency just defined and the traditional one is of "viewpoint." In the latter, the focus is *forward*, on the sequence of labels on walks leaving from a given node; in the former the focus is *backward,* on the sequences of labels on walks which terminate at a given node.

DEFINITION 4 (weak backward sense of direction). *A system* (G, λ) *has* weak backward sense of direction $\mathbf{c}$ *iff* $\mathbf{c}$ *is a backward consistent coding function. Alternatively, we shall say that* $\mathbf{c}$ *is a weak backward sense of direction in* (G, λ).

Given a coding function $\mathbf{c}$, a *backward decoding function* $\mathbf{b}$ for $\mathbf{c}$ is any function $\mathbf{b} : \mathcal{N}(\mathbf{c}) \times \Sigma \to \mathcal{N}(\mathbf{c})$ such that $\forall \pi \in P[x, y]$, $\mathbf{d}(\mathrm{a}, \mathbf{c}(\mathrm{w})) = \mathbf{c}(\mathrm{aw})$ $\langle y, z\rangle \in E(y)$, $\mathbf{b}(\mathbf{c}(\Lambda_x(\pi)), \lambda_y(\langle y, z\rangle)) = \mathbf{c}(\Lambda_x(\pi) \cdot \lambda_y(\langle y, z\rangle))$, where $\cdot$ is the concatenation operator. (Recall that $\mathcal{N}(\mathbf{c})$ denotes the codomain of $\mathbf{c}$.) We can now define backward sense of direction.

DEFINITION 5 (backward sense of direction). *A system* (G, λ) *has* backward sense of direction $(\mathbf{c}, \mathbf{b})$ *iff* $\mathbf{c}$ *is a weak backward sense of direction and* $\mathbf{b}$ *is a backward decoding function for* $\mathbf{c}$. *Alternatively, we shall say that* $(\mathbf{c}, \mathbf{b})$ *is a backward sense of direction in* (G, λ).

We shall denote by $\mathcal{W}^-$ and $\mathcal{D}^-$ the set of labeled graphs (G, λ) with backward weak sense of direction and backward sense of direction, respectively. In the following, when no ambiguity arises, we shall omit the reference to (G, λ).

3. Backward consistency.

3.1. Absence of local orientation. The first basic difference between forward and backward consistency is striking: Local orientation is *not* implied by the definition of backward sense of direction.

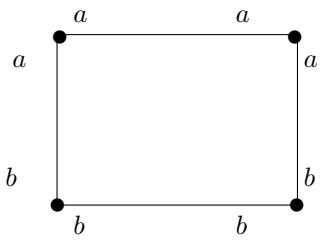


FIG. 1. $(\mathcal{D}^- - \mathcal{L}) \neq \emptyset$.

THEOREM 1 ($\mathcal{D}^- \not\subset \mathcal{L}$, $\mathcal{L} \not\subset \mathcal{D}^-$). *Local orientation is neither necessary nor sufficient for backward sense of direction.*

Proof. Consider the labeled graph (G, λ) of Figure 1. It is easy to verify that there exist both consistent backward coding and decoding functions in (G, λ). However, there is no local orientation. The fact that it is not sufficient is obvious (see, for example, the graph of Figure 4, which has local orientation but not backward sense of direction). □

Thus, in a system with backward sense of direction, nodes might not be able to distinguish among some of their incident edges. As the example in Figure 1 shows, in a system with backward sense of direction, this "blindness" can even be *complete* (i.e., it extends to all incident edges of a node) and, in the extreme case, *total* (i.e., it occurs at every node).

This phenomenon is not restricted to a few special graphs. On the contrary, *every* graph G can be labeled so as to have complete and total blindness and still have backward sense of direction.

THEOREM 2. *For any graph G there exists a labeling λ such that $\forall x \in V$, $\forall \langle x, y\rangle, \langle x, z\rangle \in E(x)$, $\lambda_x(\langle x, y\rangle) = \lambda_x(\langle x, z\rangle)$, but (G, λ) has backward sense of direction.*

Proof. Consider the labeling λ defined as follows: $\forall \langle x, y\rangle, \langle w, z\rangle \in E$, $\lambda_x(\langle x, y\rangle) = \lambda_w(\langle w, z\rangle)$ iff $x = w$ (see Figure 2). With this labeling, all the links starting from the same node x are labeled with the same label (hence, there is total and complete blindness). It is not difficult to show that the coding function $\mathbf{c}$ defined as $\mathbf{c}(a \cdot \alpha) = a$ $\forall\ a \in \Sigma, \alpha \in \Sigma^+$ is backward consistent and that the function $\mathbf{b}(\mathbf{c}(\alpha), a) = \mathbf{c}(\alpha)$ is a backward decoding for $\mathbf{c}$; hence, (G, λ) has backward sense of direction. □

Note that this is *not* the labeling used in Figure 1.

3.2. Backward local orientation. Systems with backward consistency may be without local orientation; however, they do have a property which is the "backward" analogue of local orientation. A labeling λ is a *backward local orientation* if the following holds $\forall z, \langle x, z\rangle, \langle y, z\rangle \in E(z)$: $\lambda_x(\langle x, z\rangle) \neq \lambda_y(\langle y, z\rangle)$. Let $\mathcal{L}^-$ denote the set of labeled graphs (G, λ) with backward local orientation.

First observe that backward local orientation is (obviously) *not sufficient* for backward consistency.

THEOREM 3 ($(\mathcal{L}^- - \mathcal{W}^-) \neq \emptyset$, $(\mathcal{L}^- - \mathcal{W}^-) - \mathcal{L} \neq \emptyset$). *Backward local orientation does not suffice for (either) backward consistency (or local orientation).*

Proof. Consider the labeled graph of Figure 3. Such a graph clearly has backward local orientation. By contradiction, let us assume that it has backward weak sense of direction. By definition of backward coding function we would have that, from node

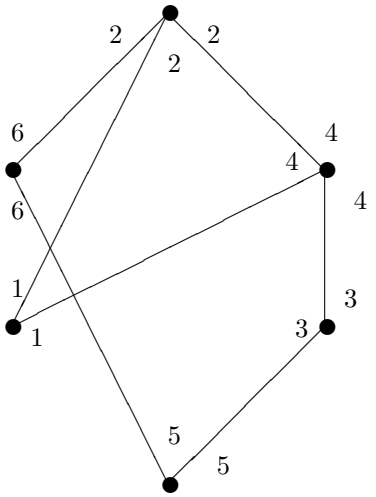


FIG. 2. *Total blindness.*

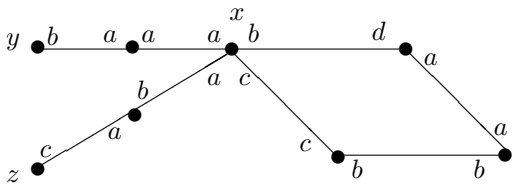


FIG. 3. $(\mathcal{L}^- - \mathcal{W}^-) - \mathcal{L} \neq \emptyset$.

x, $\mathbf{c}(b \cdot a) = \mathbf{c}(c \cdot b)$. However the labels $b \cdot a$ correspond to a path in $P[y, x]$ and $c \cdot b$ to a path in $P[z, x]$; thus, by definition of backward consistency, we must have the following: $\mathbf{c}(b \cdot a) \neq \mathbf{c}(c \cdot b)$. Since this graph does not have local orientation, this actually proves that $(\mathcal{L}^- - \mathcal{W}^-) - \mathcal{L} \neq \emptyset$. □

However, in a system with weak backward sense of direction, there is always backward local orientation.

THEOREM 4 ($\mathcal{W}^- \subset \mathcal{L}^-$). *If (G, λ) has weak backward sense of direction, then λ has backward local orientation.*

Proof. Let (G, λ) be a labeled graph without backward local orientation and with backward consistency. Then there exist two edges $\langle x, z\rangle$ and $\langle y, z\rangle$ such that $\lambda_x(\langle x, z\rangle) = \lambda_y(\langle y, z\rangle)$. But this contradicts the existence of a backward coding function $\mathbf{c}$, as by definition $\mathbf{c}(\lambda_x(\langle x, z\rangle)) = \mathbf{c}(\lambda_y(\langle y, z\rangle))$. □

Summarizing, (backward) local orientation is necessary but not sufficient for (backward) consistency. Thus, $\mathcal{W}^- \subset \mathcal{L}^-$.

Two questions naturally arise: Is the simultaneous presence of both local orientation and backward local orientation sufficient for consistency? Is it sufficient for backward consistency? The answer to these questions is negative, as we prove the stronger result that there exist labeled graphs having both forms of local orientation but without either form of consistency.

THEOREM 5 ($(\mathcal{L} \cap \mathcal{L}^-) - (\mathcal{W} \cup \mathcal{W}^-) \neq \emptyset$). *The simultaneous presence of local ori-*

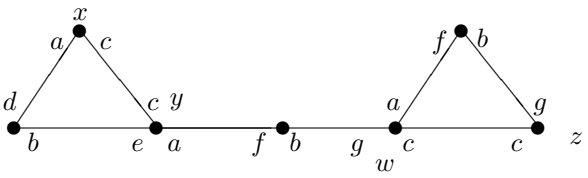


FIG. 4. $(\mathcal{L} \cap \mathcal{L}^-) - (\mathcal{W} \cup \mathcal{W}^-) \neq \emptyset$.

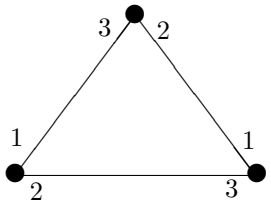


FIG. 5. $\mathcal{D} - \mathcal{L}^- \neq \emptyset$.

entation and backward local orientation does not imply either weak sense of direction or backward weak sense of direction.

Proof. Consider the labeled graph (G, λ) shown in Figure 4. The labeling clearly has both local orientation and backward local orientation. By contradiction, assume (G, λ) has weak sense of direction $\mathbf{c}$. By definition of consistent coding, at node x it must be $\mathbf{c}(a \cdot b) = \mathbf{c}(c)$; however, at node y it must be $\mathbf{c}(a \cdot b) \neq \mathbf{c}(c)$, yielding a contradiction. Assume now, again by contradiction, that (G, λ) has backward weak sense of direction $\mathbf{c}$. By definition of backward consistent coding, at node z it must be $\mathbf{c}(a \cdot b) = \mathbf{c}(c)$; however, at node w it must be $\mathbf{c}(a \cdot b) \neq \mathbf{c}(c)$, which is a contradiction. □

3.3. Orthogonality. In this section we will show that the concept of backward consistency is *orthogonal* to that of sense of direction; in particular, even the presence of sense of direction is not sufficient to guarantee the existence of a backward consistency.

THEOREM 6 ($\mathcal{D} \not\subset \mathcal{L}^-$, $\mathcal{L}^- \not\subset \mathcal{D}$). *Sense of direction is neither necessary nor sufficient for the existence of backward local orientation.*

Proof. Consider any graph with more than two nodes labeled with a *neighboring* labeling (e.g., see Figure 5); all such labeled graphs have sense of direction [20]: the coding function $\mathbf{c}$ is $\mathbf{c}(\alpha \cdot a) = a \ \forall a \in \Sigma, \alpha \in \Sigma^+$, and the decoding function is $\mathbf{d}(a, \mathbf{c}(\alpha)) = \mathbf{c}(\alpha)$. On the other hand, (G, λ) does not have backward local orientation. In other words, sense of direction is not sufficient for the existence of backward local orientation. The fact that it is also not necessary follows from Theorem 5. □

In other words, sense of direction cannot guarantee backward consistency because it cannot even guarantee backward local orientation. What if there is backward local orientation in addition to sense of direction? The answer is still negative, as shown in the following theorem.

THEOREM 7 ($(\mathcal{D} \cap \mathcal{L}^-) \not\subset \mathcal{W}^-$, $(\mathcal{D} \cap \mathcal{L}^-) \not\supset \mathcal{W}^-$). *Simultaneous presence of sense of direction and backward local orientation is neither necessary nor sufficient for the existence of a backward consistent coding function.*

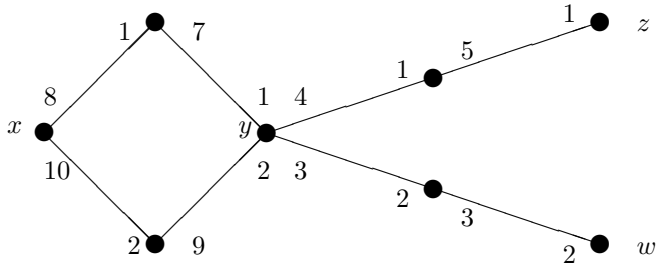


FIG. 6. $(\mathcal{D} \cap \mathcal{L}^-) \not\supset \mathcal{W}^-$.

Proof. $\not\subset$. To prove that it is not sufficient, consider the labeled graph (G, λ) of Figure 6. It is easy to verify that there exist both a consistent coding function and decoding function in (G, λ). By contradiction, assume that there exists a backward consistent coding function $\mathbf{c}$ in (G, λ). Then, by definition of backward consistency at node x we have that $\mathbf{c}(1 \cdot 1) = \mathbf{c}(2 \cdot 2)$. However, the sequence of labels $1 \cdot 1$ corresponds to a path in $P[z, y]$ and the sequence $2 \cdot 2$ to a path in $P[w, y]$; thus, by definition of backward consistency, $\mathbf{c}(1 \cdot 1) \neq \mathbf{c}(2 \cdot 2)$, yielding the contradiction.

$\not\supset$. Now we will prove that it is not necessary. By Theorem 4, backward consistency implies backward local orientation. However, from Theorem 6, it follows that backward consistency does not imply local orientation and, thus, does not imply sense of direction. This means that the simultaneous presence of sense of direction and backward local orientation is not necessary for backward consistency. □

The results of this section indicate that the presence of some additional property is necessary for one type of consistency to imply the other. We will show in the next section that *edge-symmetry* is one such property.

4. Symmetry and backward consistency. The results of the previous section indicate that the presence of some additional property is necessary for one type of consistency to imply the other. We will show that *edge-symmetry* is one such property.

Recall that a labeling λ is *symmetric* if there exists a bijection $\psi : \Sigma \to \Sigma$ such that for each $\langle x, y \rangle \in E$, $\lambda_y(\langle y, x \rangle) = \psi(\lambda_x(\langle x, y \rangle))$; ψ will be called the edge-symmetry function, and $\Psi : \Sigma^+ \to \Sigma^+$ is its extension to strings. Let $\mathcal{ES}$ denote the set of labeled graphs (G, λ) with λ symmetric.

Notice that all common labelings (e.g., "dimensional" in hypercubes, "compass" in meshes and tori, "left-right" in rings, "distance" in chordal rings, etc.) are symmetric.

4.1. Edge-symmetry and backward consistency. We will show that, while generally not true (as shown by Theorem 1), in systems with edge-symmetry, local orientation is necessary for backward consistency. In fact, we shall show that in systems with edge-symmetry, $\mathcal{L} = \mathcal{L}^-$.

THEOREM 8 ($\mathcal{ES} \cap \mathcal{L} = \mathcal{ES} \cap \mathcal{L}^-$). *Let (G, λ) be a system with edge-symmetry; then there is local orientation iff there is backward local orientation.*

Proof. Let (G, λ) be a system with edge-symmetry. $\subset$. By contradiction, suppose there is local orientation but no backward local orientation. Then there must exist two distinct edges $\langle x, z \rangle$, $\langle y, z \rangle$ such that $\lambda_x(\langle x, z \rangle) = \lambda_y(\langle y, z \rangle)$. However, by local orientation, we have that $\lambda_z(\langle z, x \rangle) \neq \lambda_z(\langle z, y \rangle)$, which contradicts the fact that there is edge-symmetry.

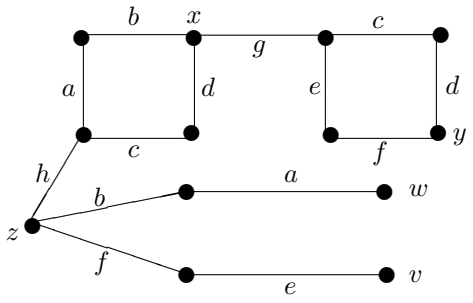


FIG. 7. $\mathcal{ES} \cap \mathcal{L} \cap \mathcal{L}^- \not\subset \mathcal{W}^-$.

$\supset$. By contradiction, suppose now that there is backward local orientation but no local orientation. Then there must exist two distinct edges $\langle x, y\rangle$, $\langle x, z\rangle$ such that $\lambda_x(\langle x, y\rangle) = \lambda_x(\langle x, z\rangle)$. However, by backward local orientation we must have $\lambda_y(\langle y, x\rangle) \neq \lambda_z(\langle z, y\rangle)$, contradicting edge-symmetry of the labeling. □

Thus, a system with edge-symmetry has either both types of local orientation or none; still, presence of both types and edge-symmetry are not sufficient for backward consistency.

THEOREM 9 ($\mathcal{ES} \cap \mathcal{L} \cap \mathcal{L}^- \not\subset \mathcal{W}^-$). *The simultaneous presence of local orientation and edge-symmetry is not sufficient for the existence of backward consistency.*

Proof. Consider the labeled graph of Figure 7. The labeling is a coloring, i.e., the edge-symmetry function is the identity function; moreover, it is a local orientation. By contradiction, assume that there exists a backward consistent coding function $\mathbf{c}$. By definition of backward consistency, at node x we have that $\mathbf{c}(a \cdot b) = \mathbf{c}(c \cdot d)$ and at node y, that $\mathbf{c}(c \cdot d) = \mathbf{c}(e \cdot f)$, which implies $\mathbf{c}(a \cdot b) = \mathbf{c}(e \cdot f)$. However, the sequence of labels $b \cdot a$ corresponds to a path in $P[w, z]$, and $f \cdot e$ corresponds to a path in $P[v, z]$; thus we have that $\mathbf{c}(a \cdot b) \neq \mathbf{c}(e \cdot f)$, which is a contradiction. □

We will now show that edge-symmetry in a system with (weak) sense of direction suffices for the system to have (weak) backward sense of direction.

LEMMA 3 (see [20]). $\forall \pi \in P[x, y]$, $\Lambda_x(\pi) = \Psi(\Lambda_y(\pi^R))$, *where* $\pi^R \in P[y, x]$ *is the reverse of walk* π.

THEOREM 10 ($\mathcal{ES} \cap \mathcal{W} \subset \mathcal{W}^-$, $\mathcal{ES} \cap \mathcal{D} \subset \mathcal{D}^-$). *Let* (G, λ) *be a system with edge-symmetry. If* (G, λ) *has weak sense of direction* $\mathbf{c}$*, then there exists a weak backward sense of direction* $\mathbf{c}_b$*. Furthermore, if* $\mathbf{c}$ *has a consistent decoding, there exists a consistent backward decoding of* $\mathbf{c}_b$*.*

Proof. Let (G, λ) have edge-symmetry and weak sense of direction $\mathbf{c}$. Let $\mathbf{c}_b$ be the coding function defined as follows: $\forall \pi \in P[x]$, $\mathbf{c}_b(\alpha) = \mathbf{c}(\Psi(\alpha))$, where $\alpha = \Lambda_x(\pi)$. We will now show that $\mathbf{c}_b$ is backward consistent.

Let $\pi_1 \in P[x, z]$, $\pi_2 \in P[y, z]$, $\alpha_1 = \Lambda_x(\pi_1)$, and $\alpha_2 = \Lambda_y(\pi_2)$. By Lemma 3, we have that $\alpha_1 = \Psi(\Lambda_z(\pi_1^R))$ and $\alpha_2 = \Psi(\Lambda_z(\pi_2^R))$. By definition of the consistent coding function, we have that $\mathbf{c}(\Psi(\Lambda_z(\pi_1^R))) = \mathbf{c}(\Psi(\Lambda_z(\pi_2^R)))$ iff $x = y$. By definition of $\mathbf{c}_b$, it follows that $\mathbf{c}_b(\Lambda_x(\pi_1)) = \mathbf{c}_b(\Lambda_y(\pi_2))$ iff $y = z$; that is, $\mathbf{c}_b$ is backward consistent.

Now let (G, λ) have edge-symmetry and sense of direction $(\mathbf{c}, \mathbf{d})$. Let $\mathbf{b}$ be the function defined as follows: $\forall \pi \in P[x, y]$, $\langle y, z\rangle \in E(y)$, $\mathbf{b}(\mathbf{c}_b(\alpha), \lambda_y(\langle y, z\rangle)) = \mathbf{d}(\psi(\lambda_y(\langle y, z\rangle)), \mathbf{c}_b(\alpha))$, where $\alpha = \Lambda_x(\pi)$. We will now show that $\mathbf{b}$ is a backward

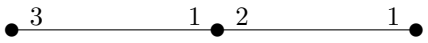


FIG. 8. $\mathcal{W} \cap \mathcal{W}^- \not\subset \mathcal{ES}$.

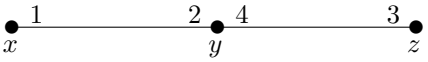


FIG. 9. *Proof of Theorem* 13.

decoding. By definition of $\mathbf{c}_b$ we have that $\mathbf{d}(\psi(\lambda_y(\langle y,z\rangle)), \mathbf{c}_b(\alpha)) = \mathbf{d}(\psi(\lambda_y(\langle y,z\rangle)), \mathbf{c}(\Psi(\alpha)))$. By definition of the decoding function we have that $\mathbf{d}(\psi(\lambda_y(\langle y,z\rangle)), \mathbf{c}(\Psi(\alpha))) = \mathbf{c}(\psi(\lambda_y(\langle y,z\rangle))\cdot\Psi(\alpha))$. But, by definition of $\mathbf{c}$, we have that $\mathbf{c}(\psi(\lambda_y(\langle y,z\rangle))\cdot\Psi(\alpha)) = \mathbf{c}_b(\alpha\cdot\lambda_y(\langle y,z\rangle))$. Thus, it follows that $\mathbf{b}(\mathbf{c}_b(\alpha), \lambda_y(\langle y,z\rangle)) = \mathbf{c}_b(\alpha\cdot\lambda_y(\langle y,z\rangle))$, proving that $\mathbf{b}$ is a backward decoding of $\mathbf{c}_b$. □

Conversely, edge-symmetry in a system with (weak) backward sense of direction suffices for the system to have (weak) sense of direction.

THEOREM 11 ($(\mathcal{ES} \cap \mathcal{W}^-) \subset \mathcal{W}$, $(\mathcal{ES} \cap \mathcal{D}^-) \subset \mathcal{D}$). *Let (G,λ) be a system with edge-symmetry. If (G,λ) has weak backward sense of direction $\mathbf{c}_b$, then there exists a weak sense of direction $\mathbf{c}$. Furthermore, if $\mathbf{c}_b$ has a consistent backward decoding, there exists a consistent decoding of $\mathbf{c}$.*

From Theorems 10 and 11, it follows that systems with edge-symmetry have either both types of consistency or none. In other words, in systems with edge-symmetry, $\mathcal{W} = \mathcal{W}^-$ and $\mathcal{D} = \mathcal{D}^-$.

An immediate question is whether systems with edge-symmetry are the only ones with such a property.

THEOREM 12 ($\mathcal{W} \cap \mathcal{W}^- \not\subset \mathcal{ES}$). *Edge-symmetry is not necessary for a system to have both forward and backward consistency.*

Proof. Consider the graph of Figure 8. This graph does not have edge-symmetry; however, it is easy to verify that there exist both a forward consistent coding function $\mathbf{c}$, and a backward consistent coding function $\mathbf{c}_b$. □

4.2. Edge-symmetry and biconsistency. In the previous section we have seen that all systems with edge-symmetry have either both types of consistency or neither. For those that do have them, the corresponding coding functions are in general different.

The question we ask now is under what circumstances a single coding function suffices, that is, when a coding function is both forward and backward consistent. We shall call any such function *biconsistent*. Let $\mathcal{W}^{\pm}$ and $\mathcal{D}^{\pm}$ denote the set of labeled graphs (G,λ) with biconsistent weak sense of direction and biconsistent sense of direction, respectively. The first result is the following.

THEOREM 13. *Edge-symmetry is not sufficient for a (backward) consistent coding function to be biconsistent.*

Proof. Consider the labeled graph in Figure 9. This labeling has local orientation. Consider any coding function such that $\mathbf{c}(1) = \mathbf{c}(3)$, $\mathbf{c}(2) \neq \mathbf{c}(4)$, $\mathbf{c}(1,2) = \mathbf{c}(1432)$, $\mathbf{c}(3,4) = \mathbf{c}(3214)$, $\mathbf{c}(2,1) = \mathbf{c}(43)$, and $\forall \alpha,\beta \in \{1,2,3,4\}^*$, $\mathbf{c}(\alpha 12\beta) = \mathbf{c}(\alpha 21\beta) = \mathbf{c}(\alpha 34\beta) = \mathbf{c}(\alpha 43\beta) = \mathbf{c}(\alpha\beta)$.

It is easy to verify that such a coding function is consistent. However, $\mathbf{c}$ is not biconsistent. In fact, $\mathbf{c}(1) = \mathbf{c}(3)$, which violates the backward consistency in y. □

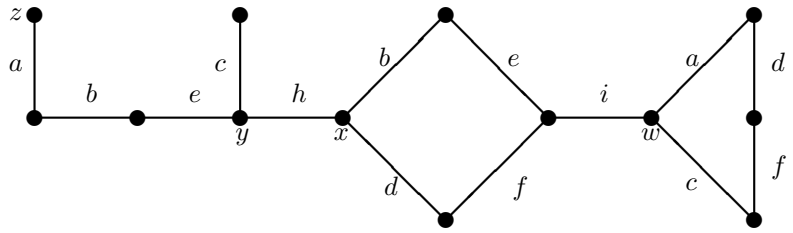


FIG. 10. $\mathcal{ES} \cap \mathcal{D}^- \not\subset \mathcal{ES} \cap \mathcal{D}$.

Even if not all the coding functions in a symmetric labeling are biconsistent, there exists at least one biconsistent coding function.

THEOREM 14 ($\mathcal{ES} \cap \mathcal{W} = \mathcal{ES} \cap \mathcal{W}^- = \mathcal{ES} \cap \mathcal{W}^{\pm}$). *In a system (G, λ) with edge-symmetry and (backward) weak sense of direction there exists also a biconsistent weak sense of direction.*

Proof. Let $\overline{\mathbf{c}}$ be a weak sense of direction in (G, λ) with edge-symmetry; we will show how to construct a weak sense of direction $\mathbf{c}$ which is biconsistent. We define the equivalence relation R between strings on the labeling alphabet in the following way: $\forall \alpha, \beta \in \Sigma^+ : \alpha R \beta \Leftrightarrow \exists x, y \in V, \pi_1, \pi_2 \in P[x, y] : \Lambda_x(\pi_1) = \alpha \wedge \Lambda_x(\pi_2) = \beta$. Let R^+ be the transitive closure of R. Let $c(\alpha) = [\alpha]_{R^+}$ (i.e., the equivalence class of the string α of the equivalence R^+).

CLAIM 1.

(i) *For all weak sense of direction* $\mathbf{c}', \alpha, \beta \in \Sigma^+ : \mathbf{c}(\alpha) = \mathbf{c}(\beta) \Rightarrow \mathbf{c}'(\alpha) = \mathbf{c}'(\beta)$.

(ii) $\forall \alpha, \beta : \mathbf{c}(\alpha) = \mathbf{c}(\beta) \Leftrightarrow \mathbf{c}(\psi(\alpha)) = \mathbf{c}(\psi(\beta))$.

(iii) $[\psi(\alpha)]_{R^+} = \psi([\alpha]_{R^+})$.

We now prove that $\mathbf{c}$ is a consistent coding function. Let $x, y, z \in V, \pi_1 \in P[x, y] \pi_2 \in P[x, z] : \Lambda_x(\pi_1) = \alpha \wedge \Lambda_x(\pi_2) = \beta$; then we have to prove that $\mathbf{c}(\alpha) = \mathbf{c}(\beta)$ iff $y = z$. By Claim 1(i), $\mathbf{c}(\alpha) = \mathbf{c}(\beta)$ implies $\overline{\mathbf{c}}(\alpha) = \overline{\mathbf{c}}(\beta)$, and by consistence of $\overline{\mathbf{c}}$ it follows that $y = z$. If $y = z$, then by definition of R it holds that $\alpha R \beta$, and then $[\alpha]_{R^+} = [\beta]_{R^+}$, that is, $\mathbf{c}(\alpha) = \mathbf{c}(\beta)$.

We now prove that $\mathbf{c}$ is a backward consistent coding function. By contradiction suppose that $\mathbf{c}$ is not consistent. Then there exists x, y, z and $\pi_1 \in P[x, z], \pi_2 \in P[y, z]$ for which $\mathbf{c}(\Lambda_x(\pi_1)) = \mathbf{c}(\Lambda(\pi_2)) \Leftrightarrow x = y$ is false. By Claim 1(i), this implies that there exists $x, y, z, \pi_1' \in P[z, x], \pi_2' \in P[z, y] : \Lambda_z(\pi_1') = \psi(\alpha)$, $\Lambda_z(\pi_2') = \psi(\beta)$, and $\mathbf{c}(\psi(\alpha)) = \mathbf{c}(\psi(\beta)) \Leftrightarrow x = y$ is false, contradicting the forward consistency of $\mathbf{c}$. □

The presence of biconsistency guaranteed by the above theorem is automatically true only for weak sense of direction. This property, unfortunately, does not extend to sense of direction, as the following theorem shows.

THEOREM 15 ($\mathcal{ES} \cap \mathcal{D} \not\subset \mathcal{W}^{\pm}$). *Sense of direction in a system with edge-symmetry is not sufficient for the existence of a biconsistent coding function.*

Proof. Consider the labeled graph of Figure 10. It is easy to verify that there exists a sense of direction for such a system. Let $(\mathbf{c}, \mathbf{d})$ be a sense of direction and suppose, by contradiction, that $\mathbf{c}$ is backward consistent.

By definition of the coding function at node x we have that $\mathbf{c}(df) = \mathbf{c}(be)$; it follows that $\mathbf{d}(a, \mathbf{c}(be)) = \mathbf{d}(a, \mathbf{c}(df)) = \mathbf{c}(c)$, which means that $\mathbf{c}(abe) = \mathbf{c}(c)$, contradicting the hypothesis of backward consistency at node y. □

As a consequence of the previous theorem we have that sense of direction in a system with edge-symmetry is not sufficient for the existence of a backward sense of direction (i.e., $\mathcal{ES} \cap \mathcal{D} \not\subset \mathcal{W}^{\pm}$).

Now we will study a sufficient property of a consistent function to be biconsistent. A weak sense of direction $\mathbf{c}$ has *name-symmetry* iff there exists a function $\mu : \mathcal{N}(\mathbf{c}) \to \mathcal{N}(\mathbf{c})$ such that $\forall \pi \in P[x,y]$, $\mu(\mathbf{c}(\Lambda_x(\pi))) = \mathbf{c}(\Lambda_y(\pi^R))$. A useful property is the following [22] lemma.

LEMMA 4 (see [22]). *Let λ be a symmetric labeling and let ψ be the corresponding function. A consistent coding function $\mathbf{c}$ has name-symmetry iff $\forall \pi_1 \in P[s,t], \pi_2 \in P[w,z]$, $\mathbf{c}(\Lambda_s(\pi_1)) = \mathbf{c}(\Lambda_w(\pi_2)) \Rightarrow \mathbf{c}(\Psi(\Lambda_s(\pi_1))) = \mathbf{c}(\Psi(\Lambda_w(\pi_2)))$.*

THEOREM 16. *In a system (G,λ) with edge-symmetry, any weak sense of direction with name-symmetry is also weak backward sense of direction.*

Proof. Let $\mathbf{c}$ be a weak sense of direction in (G,λ) with edge- and name-symmetry, and let ψ be the edge-symmetry function. By contradiction, suppose that c is not a consistent backward coding function; that is, suppose $\exists x,y,z$, $\pi_1 \in P[x,z]$, $\pi_2 \in P[y,z]$ with $x \neq y$ such that $\mathbf{c}(\Lambda_x(\pi_1)) = \mathbf{c}(\Lambda_y(\pi_2))$. Since there is name-symmetry, by Lemma 4, we have that $\mathbf{c}(\Psi(\Lambda_x(\pi_1))) = \mathbf{c}(\Psi(\Lambda_y(\pi_2)))$. But $\Lambda_z(\overline{\pi}_1) = \Psi(\Lambda_x(\pi_1))$, $\Lambda_z(\overline{\pi}_2) = \Psi(\Lambda_y(\pi_2))$ and $\overline{\pi}_1 \in P[z,x]$, $\overline{\pi}_2 \in P[z,y]$, which contradicts the consistency of $\mathbf{c}$ in z. □

In other words, in systems with edge-symmetry, any consistent coding function with name-symmetry is also backward consistent. We will now show that if any such coding function is decodable, then it is also backward decodable; that is, if there is sense of direction, there is also backward sense of direction with exactly the same coding function.

LEMMA 5. *Let $\mathbf{c}$ be a coding function with name-symmetry, and let μ be the name-symmetry function. For any $\alpha \in \Sigma^+$ corresponding to a path, $\mu(\mu(\mathbf{c}(\alpha))) = \mathbf{c}(\alpha)$.*

Proof. By definition of the name-symmetry function, we have $\mu(\mu(\mathbf{c}(\alpha))) = \mu(\mathbf{c}(\Psi(\alpha))) = \mathbf{c}(\Psi(\Psi(\alpha))) = \mathbf{c}(\alpha)$. □

THEOREM 17. *Let $(\mathbf{c},\mathbf{d})$ be a sense of direction in a system (G,λ) with edge- and name-symmetry. Then there exists a backward decoding $\mathbf{b}$ of $\mathbf{c}$; that is, $(\mathbf{c},\mathbf{b})$ is a backward sense of direction in (G,λ).*

Proof. Let ψ and μ be, respectively, the edge- and name-symmetry functions. Let $x,y \in V$, $\pi \in P[x,y]$, and $\omega = \Lambda_x(\pi)$. By definition of the name-symmetry function we have that $\mathbf{c}(\Psi(\omega)) = \mu(\mathbf{c}(\omega))$; it follows that $\mu(\mathbf{c}(\Psi(\omega))) = \mu(\mu(\mathbf{c}(\omega)))$. By Lemma 5, we have that

$$\mu(\mathbf{c}(\Psi(\omega))) = \mathbf{c}(\omega). \tag{4.1}$$

Consider now the following backward decoding function: $\forall \pi \in P[x,y]$, $\omega = \Lambda_x(\pi)$ and for $\langle y,z\rangle \in E(y) \wedge \lambda_y(\langle y,z\rangle) = a$, $\mathbf{b}(\mathbf{c}(\omega),a) = \mu(\mathbf{d}(\psi(a),\mu(\mathbf{c}(\omega))))$.

By definition of the name-symmetry function, it follows that $\mu(\mathbf{d}(\psi(a),\mu(\mathbf{c}(\omega)))) = \mu(\mathbf{d}(\psi(a),\mathbf{c}(\Psi(\omega)))) = \mu(\mathbf{c}(\Psi(w \cdot a)))$.

By (4.1), $\mu(\mathbf{c}(\Psi(w \cdot a))) = \mathbf{c}(\omega \cdot a)$. It follows that $\mathbf{b}(\mathbf{c}(\omega),a) = \mathbf{c}(\omega \cdot a)$, and thus $\mathbf{b}$ is consistent. □

5. Consistency landscape. The results of the previous sections can be seen, and have sometimes been stated, in terms of the "consistency landscape" (see Figure 11), i.e., of the relationship among the sets $\mathcal{L}, \mathcal{W}, \mathcal{D}, \mathcal{L}^-, \mathcal{W}^-, \mathcal{D}^-$.

In this section, we continue the analysis of the "consistency landscape."

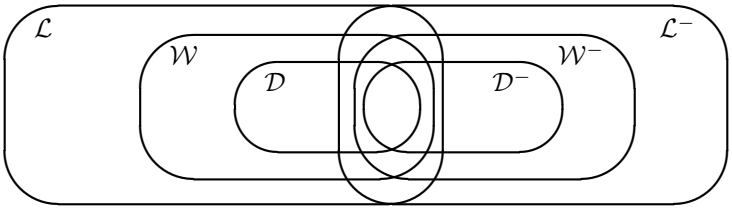


FIG. 11. *The consistency landscape.*

5.1. Transformations and constructions. In this section, we introduce two useful operations on labeled graphs. In the end, we shall derive general results on the structure of the consistency landscape.

The first operation, called "doubling," allows us to transform a labeling into a *symmetric* one. Hence, it allows us to transform a system with only one type of consistency into one which has both.

Given (G, λ), the *doubling* of λ is the mapping $\lambda_x^2((x, y))$, defined as follows: for all edges $\langle x, y\rangle$, $\lambda_x^2(\langle x, y\rangle) = (\lambda_x(\langle x, y\rangle), \lambda_y(\langle y, x\rangle))$. Its extension from edges to walks will be denoted by Λ^2.

The most important property of double labeling is that if (G, λ) has either form of consistency, then (G, λ^2) has both.

Given two strings of equal length $\alpha = a_0 a_1 \dots a_k \in \Sigma^+$ and $\beta = b_0 b_1 \dots b_k \in \Sigma^+$, let $\alpha \oplus \beta = (a_0, b_0)(a_1, b_1) \dots (a_k, b_k) \in (\Sigma^2)^+$ denote their product. Given a string $\alpha = a_0 a_1 \dots a_k \in \Sigma^+$, let $\alpha^R = a_k a_{k-1} \dots a_0$ denote the reverse string.

THEOREM 18 (double labeling). *If (G, λ) has either (weak) sense of direction or (weak) backward sense of direction, then (G, λ^2) has both (weak) sense of direction and (weak) backward sense of direction.*

Proof. Let $\mathbf{c}$ be a coding function in (G, λ). Consider the coding function $\mathbf{c}_2$ defined as follows: $\forall\, \alpha \oplus \beta \in (\Sigma^2)^+$, $\mathbf{c}_2(\alpha \oplus \beta) = \mathbf{c}(\alpha)$. Clearly, $\mathbf{c}_2$ is (resp., backward) consistent in (G, λ^2) iff $\mathbf{c}$ is (resp., backward) consistent in (G, λ).

Let $\mathbf{c}$, and hence $\mathbf{c}_2$, be (resp., backward) consistent. Given a function $\mathbf{d} : \Sigma \times \mathcal{N}(\mathbf{c}) \to \mathcal{N}(\mathbf{c})$, define the function $\mathbf{d}_2$ as follows: $\forall\ (a, b) \in \Sigma^2$ and $\alpha \oplus \beta \in (\Sigma^2)^+$, $\mathbf{d}_2((a, b), \mathbf{c}_2(\alpha \oplus \beta)) = \mathbf{d}(a, \mathbf{c}(\alpha))$. Clearly, $\mathbf{d}_2$ is a decoding of $\mathbf{c}_2$ iff $\mathbf{d}$ is a decoding of $\mathbf{c}$. Similarly, given $\mathbf{b} : \mathcal{N}(\mathbf{c}) \times \Sigma \to \mathcal{N}(\mathbf{c})$, the function $\mathbf{b}_2(\mathbf{c}_2(\alpha \oplus \beta), (a, b)) = \mathbf{d}(a, \mathbf{c}(\alpha))$ is a backward decoding of c_2 iff $\mathbf{b}$ is a backward decoding of $\mathbf{c}$.

In other words, (weak) sense of direction in (G, λ) implies (weak) sense of direction in (G, λ^2), and (weak) backward sense of direction in (G, λ) implies (weak) backward sense of direction in (G, λ^2).

Finally observe that λ^2 is symmetric; in fact, for every $\langle x, y\rangle \in E$, $\lambda_x^2(\langle x, y\rangle) = \lambda_y^2(\langle y, x\rangle)^R$. Thus, by Theorems 10 and 11, if (G, λ^2) has one type of consistency, it has both. □

The operation of doubling is important in that it allows us to transform a labeling into a *symmetric* one, and thus to extend the existing consistency to include also the other type.

The proof of Theorem 18 shows also how to employ the (resp., backward) coding and decoding of the original system to construct the analogous ones of the new system. However, it does not give any indication on the nature of the coding and decoding of the opposite type. As we will see later, such a nature is useful also for the other

transformation to be discussed in this section.

The purpose of the following lemmas is to show how to construct in the new systems the coding and decoding of the opposite type.

LEMMA 6. *Let* $\mathbf{c}$ *be a weak sense of direction in* (G,λ)*; then* $\mathbf{c}_b(\alpha \oplus \beta) = \mathbf{c}(\beta^R)$ *is a weak backward sense of direction in* (G,λ^2)*. Furthermore, if* $\mathbf{c}$ *has a decoding* $\mathbf{d}$*, then* $\mathbf{b}(\mathbf{c}_b(\alpha \oplus \beta),(a,b)) = \mathbf{d}(b,\mathbf{c}(\beta^R))$ *is a backward decoding of* $\mathbf{c}_b$*.*

Proof. Let $x,y,z \in V$, $\pi_1 \in P[x,z]$, $\pi_2 \in P[y,z]$. Let $\Lambda_x^2(\pi_1) = (a_0,a_0')\ (a_1,a_1')\ \dots\ (a_h,a_h')$, and $\Lambda_y^2(\pi_2) = (b_0,b_0')(b_1,b_1')\dots(b_k,b_k')$.

Let $x = y$. By definition of $\mathbf{c}_b$ we have that $\mathbf{c}_b(\Lambda_x^2(\pi_1)) = \mathbf{c}(a_h' \dots a_0') = \mathbf{c}(\Lambda_z(\overline{\pi_1})$. Since $x = y$, by definition of the consistency of $\mathbf{c}$ we have that $\mathbf{c}(\Lambda_z({\pi_1}^R)) = \mathbf{c}(\Lambda_z({\pi_2}^R)) = \mathbf{c}(b_k' \dots b_0')$. Since $\mathbf{c}(b_k' \dots b_0') = \mathbf{c}_b(\Lambda_y^2(\pi_2))$, it follows that $\mathbf{c}_b(\Lambda_x^2(\pi_1)) = \mathbf{c}_b(\Lambda_y^2(\pi_2))$.

Let $x \neq y$. By definition of $\mathbf{c}_b$ we have that $\mathbf{c}_b(\Lambda_x^2(\pi_1)) = \mathbf{c}(a_h' \dots a_0')$. Since $x \neq y$, by definition of the consistency of $\mathbf{c}$ we have that $\mathbf{c}(\Lambda_z({\pi_1}^R)) \neq \mathbf{c}(\Lambda_z({\pi_2}^R))$. But $\Lambda_z({\pi_1}^R) = (a_h' \dots a_0')$, and $\Lambda_z({\pi_2}^R) = b_k' \dots b_0'$. Since $\mathbf{c}(b_k' \dots b_0') = \mathbf{c}_b(\Lambda_y^2(\pi_2))$, it follows that $\mathbf{c}_b(\Lambda_x^2(\pi_1)) \neq \mathbf{c}_b(\Lambda_y(\pi_2))$.

We now show that the corresponding backward consistent decoding function is the following: $\forall \pi \in P[x,y]$, $\forall \langle y,z\rangle \in E(y)$, let $\Lambda_x^2(\pi) = (\alpha \oplus \beta))$ and $\lambda_y^2(\langle y,z\rangle) = (a,b)$; then $\mathbf{b}(\mathbf{c}_b(\alpha \oplus \beta),(a,b)) = \mathbf{d}(b,\mathbf{c}(\beta^R)$.

By definition of the consistent decoding function, $\mathbf{d}(b,\mathbf{c}(\beta^R) = \mathbf{c}(b \cdot \beta^R)$. By definition of $\mathbf{c}_b$ we have that $\mathbf{c}(b \cdot \beta^R) = \mathbf{c}_b((\alpha \oplus \beta) \cdot (a,b))$. Thus, it follows that $\mathbf{b}$ is consistent, i.e., $\mathbf{b}(\mathbf{c}_b(\alpha \oplus \beta),(a,b)) = \mathbf{c}_b((\alpha \oplus \beta) \cdot (a,b))$. □

Similarly, we have the following lemma.

LEMMA 7. *Let* $\mathbf{c}$ *be a weak backward sense of direction in* (G,λ)*. Then* $\mathbf{c}_f(\alpha \oplus \beta) = \mathbf{c}(\beta^R)$ *is a weak sense of direction in* (G,λ^2)*. Furthermore, if* $\mathbf{c}$ *has a backward decoding* $\mathbf{b}$*, then* $\mathbf{d}((a,b),\mathbf{c}_f(\alpha \oplus \beta)) = \mathbf{b}(\mathbf{c}(\beta^R),b)$ *is a decoding of* $\mathbf{c}_f$*.*

The second transformation is "reversal." Given a labeling λ of a graph G, the *reverse labeling* $\tilde{\lambda}$ of G is obtained in the following way: $\forall \langle x,y\rangle \in E$, $\tilde{\lambda}_x(\langle x,y\rangle) = \lambda_y(\langle y,x\rangle)$.

The reversal operation is related to doubling as follows, where $\mathbf{c}_b$, $\mathbf{c}_f$, $\mathbf{b}$, and $\mathbf{d}$ are as defined in Lemmas 6 and 7.

LEMMA 8. *Let* $\mathbf{c}$ *be a weak sense of direction in* (G,λ)*; then* $\mathbf{c}_b$ *is a weak backward sense of direction in* $(G,\tilde{\lambda})$*. Furthermore, if* $\mathbf{c}$ *has a decoding* $\mathbf{d}$*, then* $(\mathbf{c}_b,\mathbf{b})$ *is backward sense of direction in* $(G,\tilde{\lambda})$*.*

LEMMA 9. *Let* $\mathbf{c}$ *be a weak backward sense of direction in* (G,λ)*; then* $\mathbf{c}_f$ *is a weak sense of direction in* $(G,\tilde{\lambda})$*. Furthermore, if* $\mathbf{c}$ *has a backward decoding* $\mathbf{b}$*, then* $(\mathbf{c}_f,\mathbf{d})$ *is a sense of direction in* $(G,\tilde{\lambda})$*.*

Proof. It directly follows from Theorem 18 that the following coding function is consistent: $\forall \pi \in P[x_0]$, $\pi = \langle x_0,x_1\rangle \dots \langle x_{m-1},x_m\rangle$: $\mathbf{c}_b(\tilde{\Lambda}_x(\pi)) = \mathbf{c}(\tilde{\lambda}_{x_{m-1}}(\langle x_{m-1},x_m\rangle)\ \dots\ \tilde{\lambda}_{x_0}(\langle x_0,x_1\rangle))$. The corresponding backward decoding function is the following: $\forall \pi \in P[x_0]$, $\pi = (\langle x_0,x_1\rangle \dots \langle x_{m-1},x_m\rangle)$, $\forall \langle x_m,y\rangle \in E(x_m)$: $\mathbf{b}(\mathbf{c}_b(\tilde{\Lambda}_{x_0}(\pi)),\ \tilde{\lambda}_{x_m}(\langle x_m,y\rangle)) = \mathbf{d}(\tilde{\lambda}_{x_m}(\langle x_m,y\rangle),\ \mathbf{c}(\tilde{\lambda}_{x_{m-1}}(\langle x_{m-1},x_m\rangle)\ \dots\ \tilde{\lambda}_{x_0}(\langle x_0,x_1\rangle)))$. □

The above lemmas have a very important consequence, as follows.

THEOREM 19. (G,λ) *has (weak) backward sense of direction iff* $(G,\tilde{\lambda})$ *has (weak) sense of driection.*

This result implies that the structure of the "forward" consistency landscape and that of the backward consistent are mirror images. This fact will be instrumental in simplifying the proof of some of the results to be established in the remainder of this section.

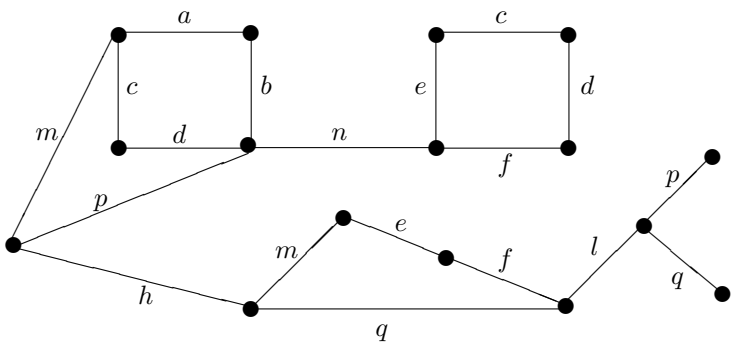


FIG. 12. $\mathbf{G}_w$.

5.2. Core. We now focus on the structure of the "core" of the consistency landscape; i.e., the set $\mathcal{W} \cap \mathcal{W}^-$.

The first result we establish is that backward sense of direction is a stronger form of consistency than weak backward sense of direction.

THEOREM 20 ($\mathcal{D}^- \subset \mathcal{W}^-$). *There are systems with backward consistency where no coding function is backward decodable.*

Proof. The proof follows from Lemmas 2 and 9. □

The next question we consider is whether the simultaneous presence of both consistencies suffices for the existence of either form of decoding. The answer is negative, as there exist systems with both weak sense of direction and weak backward sense of direction where no (backward) coding function is (backward) decodable. Let $\mathbf{G}_w$ be the labeled graph shown in Figure 12.

LEMMA 10 (see [11]). $\mathbf{G}_w \in \mathcal{W} - \mathcal{D}$.

THEOREM 21 ($(\mathcal{W} \cap \mathcal{W}^-) \not\subset (\mathcal{D} \cup \mathcal{D}^-)$). *Existence of both types of consistency is not sufficient for decodability of either type.*

Proof. By Lemma 10, graph G_w has weak sense of direction, but any coding function is not decodable. Such a labeled graph has edge-symmetry since the labeling is a coloring. Thus, by Theorem 10 we have that it also has backward weak sense of direction. However, no backward consistent coding function for G_w can be backward decodable; otherwise G_w would have backward sense of direction and, thus, by Theorem 11 would also have sense of direction contradicting Lemma 10. □

THEOREM 22 ($(\mathcal{D} \cap \mathcal{W}^-) \not\subset \mathcal{D}^-$, $\mathcal{D}^- \not\subset (\mathcal{W}^- \cap \mathcal{D})$). *Simultaneous presence of sense of direction and backward weak sense of direction is neither necessary nor sufficient for the existence of a backward decoding function.*

Proof. Not sufficient. It is easy to verify that the labeled graph of Figure 13 has sense of direction and weak backward sense of direction. However, no backward coding function is backward decodable. In fact, let $\mathbf{c}$ be a backward coding function; by definition, we have that $\mathbf{c}(dc) = \mathbf{c}(fe)$ and $\mathbf{c}(ba) = \mathbf{c}(dc)$; it follows that $\mathbf{c}(ba) = \mathbf{c}(fe)$. Let $\mathbf{b}$ be a backward decoding function; then we have that $\mathbf{b}(\mathbf{c}(ba), m) = \mathbf{c}(p) = \mathbf{b}(\mathbf{c}(fe), m)$. But $\mathbf{b}(\mathbf{c}(fe), m) = \mathbf{c}(q)$. It follows that $\mathbf{c}(q) = \mathbf{c}(p)$, which contradicts both forward and backward coding.

Not necessary. It follows from the fact that local orientation is not necessary for having a backward decoding function (see Theorem 1). □

Thus, by Theorem 19, the following holds.

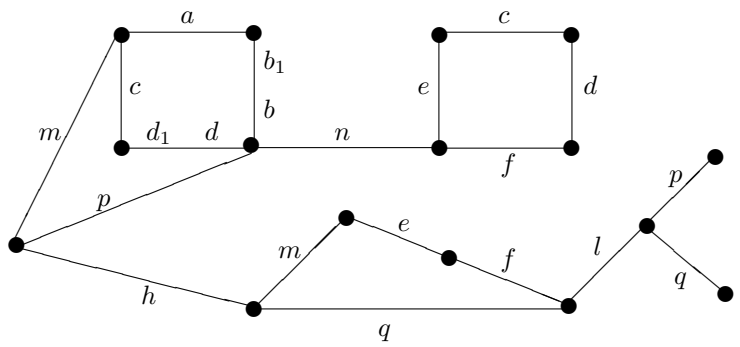


FIG. 13. $(\mathcal{D} \cap \mathcal{W}^-) - \mathcal{D}^- \neq \emptyset$.

THEOREM 23 ($(\mathcal{D}^- \cap \mathcal{W}) \not\subset \mathcal{D}$). *Simultaneous presence of backward sense of direction and weak sense of direction is not sufficient for the existence of a decoding function.*

5.3. Outer structure. In this section we focus on the outer structure of the consistency landscape: $(\mathcal{L} \cup \mathcal{L}^-) - (\mathcal{W} \cap \mathcal{W}^-)$.

We first introduce two useful properties.

Given two labeled graphs (G_1, λ_1) and (G_2, λ_2), the *melding* of (G_1, λ_1) and (G_2, λ_2) by $x_1 \in V_1, x_2 \in V_2$, denoted by $\mathbf{G_1}[x_1, x_2]\mathbf{G_2}$, is the union of the two graphs restricted by imposing $x_1 = x_2$.

THEOREM 24. *Let $\mathbf{G_1}$ and $\mathbf{G_2}$ be vertex- and label-disjoint labeled graphs with weak sense of direction. Then $\forall x_1 \in V_1, x_2 \in V_2$, $\mathbf{G} = \mathbf{G_1}[x_1, x_2]\mathbf{G_2}$ also has weak sense of direction. Furthermore, if $\mathbf{G_1}$ and $\mathbf{G_2}$ have sense of direction, (G, λ) also has sense of direction.*

Proof. By hypothesis the labeled graphs are label-disjoint, that is, $\Sigma_1 \cap \Sigma_2 = \emptyset$. By hypothesis there exists two coding function $\mathbf{c}_1$ and $\mathbf{c}_2$ such that $\mathbf{c}_1$ is a consistent coding function for G_1 and $\mathbf{c}_2$ for G_2. Without loss of generality we can also assume that $\mathcal{N}(\mathbf{c}_1) \cap \mathcal{N}(\mathbf{c}_2) = \emptyset$. (If this is not the case, take $\mathbf{c}'_1 = (0, \mathbf{c}_1)$ and $\mathbf{c}'_2 = (1, \mathbf{c}_2)$.) Let $i : (\Sigma_1 \cup \Sigma_2)^+ \to \{1, 2\}$ be a function such that $i(\alpha) = 1$ iff the last symbol of α belongs to Σ_1; otherwise, $i(\alpha) = 2$. Let $f : (\Sigma_1 \cup \Sigma_2)^+ \to \Sigma_1^+ \cup \Sigma_2^+$ be a function such that $f(\alpha) = \alpha \cap \Sigma_{i(\alpha)}$ (i.e., $f(\alpha)$ is the string of all the symbols of α belonging to the set $\Sigma_{i(\alpha)}$).

Let $p(x, \alpha) = x$ if $x \in G_{i(\alpha)}$, and $x_1 = x_2$ otherwise. Then it is easy to see that

$$(*) \qquad \forall \alpha \in \Lambda_G[x, y] \Rightarrow f(\alpha) \in \Lambda_{G_{i(\alpha)}}[p(x, \alpha), y].$$

Now let $\mathbf{c}$ be so defined: $\mathbf{c}(\alpha) = \mathbf{c}_{i(\alpha)}(f(\alpha))$. Now we will prove that $\mathbf{c}$ is consistent. By contradiction suppose that $\mathbf{c}$ is not consistent; then there exist $x, y, z \in V_G$ and $\alpha \in \Lambda_G[x, y], \beta \in \Lambda_G[x, z]$ such that either $\mathbf{c}(\alpha) = \mathbf{c}(\beta)$ but $y \neq z$, or $\mathbf{c}(\alpha) \neq \mathbf{c}(\beta)$ but $y = z$.

In the first case, since the codomains of $\mathbf{c}_1$ and $\mathbf{c}_2$ are disjoint, it follows that $\mathbf{c}_i(f(\alpha)) = \mathbf{c}_i(f(\beta))$ for some i such that $f(\alpha), f(\beta) \in \Sigma_i^+$. By (*) it follows that $f(\alpha) \in \Lambda_{G_i}[p(x, \alpha), y]$ and $f(\beta) \in \Lambda_{G_i}[p(x, \beta), z]$. Because of $i(\alpha) = i(\beta)$ it follows that $p(x, \alpha) = p(x, \beta)$. Thus, $f(\alpha) \in \Lambda_{G_i}[x', y]$ and $f(\beta) \in \Lambda_{G_i}[x', z]$. By consistency of c_i it follows that $y = z$, yielding a contradiction.

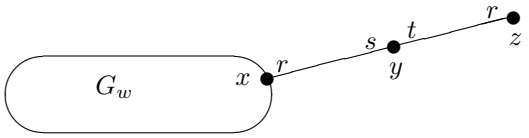


FIG. 14. $(\mathcal{W} - \mathcal{D}) - \mathcal{L}^- \neq \emptyset$.

In the second case, if $y = z$ and $y \in G_i$ for some i, by (*) and analogous deductions, we have that $f(\alpha), f(\beta) \in \Lambda_{G_i}[x', y]$ and, by consistency of $\mathbf{c}_i$, $\mathbf{c}_i(f(\alpha)) = \mathbf{c}_i(f(\beta))$; that is, $\mathbf{c}(\alpha) = \mathbf{c}(\beta)$, yielding a contradiction.

Thus, we have proved that $\mathbf{c}$ is consistent; that is, G has weak sense of direction.

Now we will prove that if G_1 and G_2 have sense of direction, then G also has sense of direction. Let $\mathbf{d}_1$ and $\mathbf{d}_2$ be the decoding functions, and let $\mathbf{d}$ be so defined:

$$\mathbf{d}(a, \mathbf{c}(w)) = \begin{cases} \mathbf{d}_1(a, \mathbf{c}(w)) & \text{if } a \in \Sigma_1 \wedge \mathbf{c}(w) \in \mathcal{N}(\mathbf{c}_1) \quad (1) \\ \mathbf{d}_2(a, \mathbf{c}(w)) & \text{if } a \in \Sigma_2 \wedge \mathbf{c}(w) \in \mathcal{N}(\mathbf{c}_2) \quad (2) \\ \mathbf{c}(w) & \text{if } a \in \Sigma_1 \wedge \mathbf{c}(w) \in \mathcal{N}(\mathbf{c}_2) \quad (3) \\ \mathbf{c}(w) & \text{if } a \in \Sigma_2 \wedge \mathbf{c}(w) \in \mathcal{N}(\mathbf{c}_1) \quad (4) \end{cases}.$$

We will show that $\mathbf{d}(a, \mathbf{c}(w)) = \mathbf{c}(aw)\ \forall\ aw \in \Lambda_G[x, z]$. We have to consider the four cases in the definition of $\mathbf{d}$.

(1) In this case $\mathbf{d}(a, \mathbf{c}(w)) = \mathbf{d}_1(a, \mathbf{c}(w))$, $a \in \Sigma_1$, and $\mathbf{c}(w) \in \mathcal{N}(\mathbf{c}_1)$. Thus, a is a label of an edge of G_1 and w is a label of a path ending in G_1. This means that $\mathbf{c}(w) = \mathbf{c}_1(f(w))$. Then, $\mathbf{d}_1(a, \mathbf{c}(w)) = \mathbf{d}_1(a, \mathbf{c}_1(f(w)))$. By property (*), it follows that $f(w)$ is the label of a path in G_1; since $x \in G_1$, $a \cdot f(w) = f(aw) \in \Lambda_{G_1}[x, z]$. By consistency of $\mathbf{d}_1$, we have $\mathbf{d}_1(a, \mathbf{c}_1(f(w))) = \mathbf{c}_1(a \cdot f(w)) = \mathbf{c}_1(f(aw)) = \mathbf{c}(aw)$.

(2) Symmetric to case (1).

(3) In this case, $\mathbf{d}(a, \mathbf{c}(w)) = \mathbf{c}(w)$, $a \in \Sigma_1$, and $\mathbf{c}(w) \in \mathcal{N}(\mathbf{c}_2)$. Thus, a is a label of an edge in G_1, while w is the labeling of a path ending in G_2. This implies that $f(aw) = f(w)$. Thus, $\mathbf{d}(a, \mathbf{c}(w)) = \mathbf{c}(w) = \mathbf{c}_2(f(w)) = \mathbf{c}_2(f(aw)) = \mathbf{c}(aw)$.

(4) Symmetric to case (3). □

THEOREM 25. $(\mathcal{W} - \mathcal{D}) \not\subset \mathcal{L}^-$.

Proof. Consider the labeled graph of Figure 14. Notice that such a graph is the melding in x of the graph G_w of Figure 12 and a line with two edges $\langle x, y\rangle$ and $\langle y, z\rangle$.

Since the labeled line has trivially weak sense of direction and since, by Lemma 10, the graph $\mathbf{G}_w$ of Figure 12 has weak sense of direction, it follows from Theorem 24 that the graph of Figure 14 has weak sense of direction as well.

On the other hand, by Lemma 10, the graph of Figure 12 does not have sense of direction, and, as a consequence, the graph of Figure 14 also does not have sense of direction. Moreover, this labeled graph does not have backward local orientation; in fact $\lambda_x(\langle x, y\rangle) = \lambda_z(\langle z, y\rangle)$. □

As a consequence, we have the following.

THEOREM 26. $(\mathcal{W}^- - \mathcal{D}^-) - \mathcal{L} \neq \emptyset$.

Proof. The proof follows from Theorems 19 and 25. □

THEOREM 27. $((\mathcal{W} - \mathcal{D}) \cap \mathcal{L}^-) - \mathcal{W}^- \neq \emptyset$.

Proof. Consider the labeled graph (G, λ) of Figure 15. Notice that it is the melding in x of Figure 12 and a labeled graph (G', λ'). It is easy to verify that (G', λ') has weak sense of direction, which implies that (G, λ) has weak sense of direction. However,

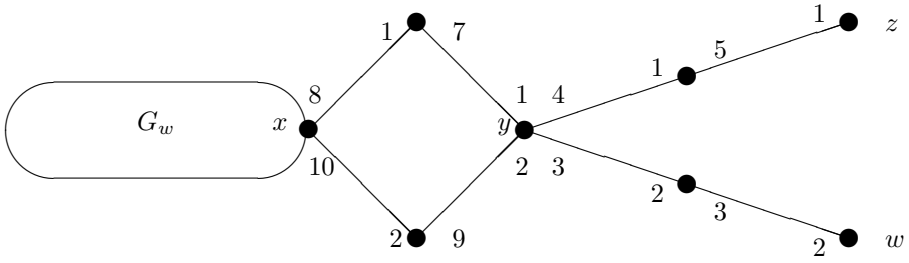


FIG. 15. $((\mathcal{W} - \mathcal{D}) \cap \mathcal{L}^-) - \mathcal{W}^- \neq \emptyset$.

since by Lemma 10 the graph G_w of Figure 12 does not have sense of direction, it follows that (G, λ) also does not have sense of direction. Clearly, there is backward local orientation. By contradiction, let us assume that there is weak backward sense of direction in (G, λ). By definition of backward consistency at node x we would have that $\mathbf{c}(1 \cdot 1) = \mathbf{c}(2 \cdot 2)$ (since these two sequences of labels correspond to two paths from y to x); however, $\mathbf{c}(1 \cdot 1) \neq \mathbf{c}(2 \cdot 2)$ (since they also correspond to two paths starting from different nodes (w and z) and terminating in the same node y). □

As a consequence, we have the following.

THEOREM 28. $((\mathcal{W}^- - \mathcal{D}^-) \cap \mathcal{L}) - \mathcal{W} \neq \emptyset$.

Proof. The proof follows from Theorems 19 and 27. □

6. Backward consistency and Σ. In this section we ask some questions about the relationship between the set of labels used in (G, λ) and the existence of backward consistency. In particular, we focus on the number $\kappa(G, \lambda) = |\Sigma|$ of labels used in (G, λ). A general bound on κ is the following, where $\deg(G)$ and $e(G)$ denote the maximum degree and the number of edges in G, respectively.

LEMMA 11. *If (G, λ) has backward consistency, then $d(G) \leq \kappa(G, \lambda) \leq 2e(G)$.*

6.1. Minimum backward sense of direction. If $\kappa(G, \lambda) = d(G)$, the labeling is said to be *minimum* and so is called any consistency property existing in (G, λ). So, consistency in minimally labeled systems is called *minimum (weak) sense of direction*; similarly we can define the backward analogues. Let $m\mathcal{C}$ denote the subclass of $\mathcal{C}$ of labeled graphs (G, λ) for which λ is a minimum labeling (e.g., $m\mathcal{W}$ is the class of labeled graphs with minimum weak sense of direction).

We now study the relationship between minimality and biconsistency in arbitrary topologies.

THEOREM 29. *Edge-symmetry in a minimum weak sense of direction is not sufficient for backward consistency.*

Proof. Consider the labeled graph of Figure 16.

For each $\alpha \in \{r, l, a\}^*$, let $\mu(\alpha) \in \{r, a\}^*$, the string obtained from α applying as many times as possible one of the following simplification rules: $rl \to \epsilon$, $lr \to \epsilon$, $aa \to \epsilon$, $rrr \to \epsilon$, $lll \to \epsilon$, $l \to rr$. The first three rules eliminate all the cycles of length 2, and the following two rules eliminate all the cycles of length 3. The last rule rewrites the l with two r's. Let $A = \{\alpha : \exists v \wedge \exists \pi \in P[v] : \Lambda_v(\pi) = \alpha\}$, and let $\mu(A) = \{\mu(\alpha) : \alpha \in A\}$. It can be proven that $\mu(A) = r^{2*}(\cdot a \cdot r^{2*})^?$, where r^{2*} means at most 2 repetitions of the symbol r (there can be 0 repetitions). $(x)^?$ means 0 or 1 repetition of x. Let a coding function c be defined as follows:

$$c(\alpha) = \begin{cases} (\mu(\alpha), 0) & \text{if } \mu(\alpha) \in r^{2*}, \\ (\beta, 1)) & \text{if } \mu(\alpha) = r^{2*}a\beta. \end{cases}$$

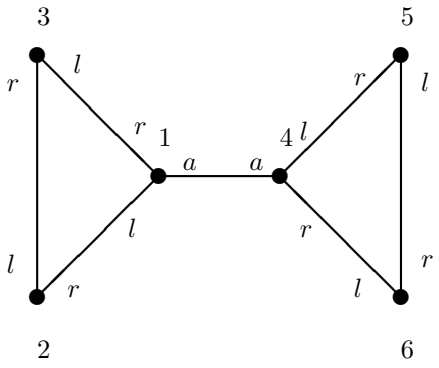


FIG. 16. *A minimum labeled graph with a consistent coding c that is not backward consistent.*

For each node v let $\overline{v}$ be the node on the same ring with degree 3 (e.g., $\overline{2} = \overline{3} = \overline{1} = 1$ and $\overline{4} = \overline{5} = \overline{6} = 4$). We can prove the following property.

CLAIM 2.

1. $\forall x, y, \pi \in P[x, y], (\mathbf{c}(\Lambda_x(\pi)) = \mathbf{c}(\gamma, 0) \Leftrightarrow \overline{x} = \overline{y} \wedge \exists \pi' \in P[x, y] \wedge \Lambda_x(\pi') = \gamma)$.
2. $\forall x, y, \pi \in P[x, y], (\mathbf{c}(\Lambda_x(\pi)) = \mathbf{c}(\gamma, 1) \Leftrightarrow \overline{x} \neq \overline{y} \wedge \exists \pi' \in P[\overline{y}, y] \wedge \Lambda_{\overline{y}}(\pi') = \gamma)$.

It is easy to see that $\mathbf{c}$ is a consistent coding function. But $\mathbf{c}$ is not backward consistent. We can see that ra is the label of the path from 6 to 1 passing from node 4, and a is the label of the path from 4 to 1. Thus $4 \neq 6$, but $\mathbf{c}(ra) = (\epsilon, 1) = \mathbf{c}(a)$. □

COROLLARY 1. *Edge-symmetry in a minimum (backward) weak sense of direction is not sufficient for biconsistency.*

Note that this also gives, as a corollary, a different proof of Theorem 13.

Now we will study the relationship between sense of direction and minimality in regular graphs. Let $\mathcal{R}$ denote the set of labeled regular graphs, that is, $\deg(v) = k$ for all nodes v.

First observe that the requirement of minimality does not make edge-symmetry necessary for either form of consistency.

LEMMA 12 ($\mathcal{R} \cap m\mathcal{W}^{(-)} \not\subset \mathcal{ES}$). *Edge-symmetry is not necessary for having minimum weak sense of direction or minimum weak backward sense of direction in regular graphs.*

Proof. The nonnecessity for minimum weak backward sense of direction can be seen by considering Figure 1, where the graph has minimum weak backward sense of direction but does not have edge-symmetry; the mirror image result follows by Theorem 19. □

However, for the simultaneous existence of both types of consistency, unlike the general case (see Theorem 12), edge-symmetry is necessary if minimality is required.

THEOREM 30 ($m\mathcal{W}^{\pm} \cap \mathcal{R} = m\mathcal{W} \cap \mathcal{ES} \cap \mathcal{R} = m\mathcal{W}^{-} \cap \mathcal{ES} \cap \mathcal{R}$). *In a regular labeled graph (G, λ) with edge-symmetry, any minimum weak sense of direction (minimum weak backward sense of direction) is also biconsistent.*

Proof. Let $\mathbf{c}$ be a consistent coding function for (G, λ). By contradiction, let us assume that $\mathbf{c}$ does not have backward consistency, which means that there exist two paths $\pi_1 \in P[x, z], \pi_2 \in P[y, z]$ such that either (i) $\mathbf{c}(\Lambda_x(\pi_1)) = \mathbf{c}(\Lambda_y(\pi_2))$ and $x \neq y$ or (ii) $\mathbf{c}(\Lambda_x(\pi_1)) \neq \mathbf{c}(\Lambda_y(\pi_2))$ and $x = y$.

Let us consider case (i). Since the graph is regular and the labeling minimum, there must exist a path $\pi_3 \in P[x]$ such that $\Lambda_x(\pi_3) = \Lambda_y(\pi_2)$. Thus, $\mathbf{c}(\Lambda_x(\pi_1)) =$

$\mathbf{c}(\Lambda_y(\pi_3))$, which implies that $\pi_3 \in P[x,z]$. This is a contradiction, because it violates local orientation in z.

Consider now case (ii). This case violates the definition of forward consistency in x.

The reverse is analogous. □

Thus, with edge-symmetry, the class of regular graphs with minimum weak backward sense of direction coincides with the class of edge-symmetric regular graphs with minimum weak sense of direction. Using the characterization of the latter class by [10, 23], we obtain the following.

THEOREM 31. *A regular labeled graph (G,λ) with edge-symmetry has minimum weak backward sense of direction iff it is a Cayley graph with Cayley labeling.*

In other words, if there is edge-symmetry, Cayley graphs with Cayley labelings are the only regular graphs with minimum backward sense of direction and/or minimum sense of direction.

COROLLARY 2 ($m\mathcal{W}^{\pm} \cap \mathcal{ES} \cap \mathcal{R} = m\mathcal{D}^{\pm} \cap \mathcal{ES} \cap \mathcal{R}$). *In a regular labeled graph with edge-symmetry, any minimum and biconsistent weak sense of direction is a sense of direction.*

The situation is quite different in the absence of edge-symmetry; in this case, at most one type of minimum consistency can exist.

LEMMA 13 ($m\mathcal{W} \cap m\mathcal{W}^{-} \subset \mathcal{ES}$). *Without edge-symmetry and restricted to regular graphs, $m\mathcal{W} \cap m\mathcal{W}^{-} = \emptyset$.*

Proof. The proof follows from Theorem 30. □

6.2. Number of labels and backward consistency. In this section we ask under what conditions a (large) value $\kappa(G,\lambda)$ is sufficient for backward consistency. The first obvious result is that $\kappa(G,\lambda) = 2e(G)$; then the system has both sense of direction and backward sense of direction without further conditions.

LEMMA 14. *If $\kappa(G,\lambda) = 2e(G)$, then (G,λ) has both sense of direction and backward sense of direction.*

Basically, this numerical requirement is equivalent to having all edges labeled with unique labels.

If we are interested only in backward sense of direction, we can actually relax this condition. In fact, it is sufficient that the labels used by each node are different from the ones used by the other nodes for the system to have backward sense of direction (but not necessarily sense of direction); see, for example, Figure 17. Let $\Sigma_x = \{\lambda_x(x,y) : y \in E(x)\}$.

THEOREM 32. *If, in (G,λ), $\Sigma_x \cap \Sigma_y = \emptyset \ \forall\ x \neq y$, then (G,λ) has backward sense of direction.*

Proof. For each $a \in \Sigma$, only one node x uses a: let $\sigma(a) = \Sigma_x$.

Consider the following backward coding function: $\forall \pi \in P[x_0]$, $\pi = \langle x_0, x_1\rangle \dots \langle x_{m-1}, x_m\rangle$, $\mathbf{c}(\Lambda_x(\pi)) = \sigma(\lambda_{x_0}(\langle x_0, x_1\rangle))$.

Let $\pi_1 \in P[x,z], \pi_2 \in P[y,z]$; $\pi_1 = \langle x, x_1\rangle \dots \langle x_h, z\rangle$, $\pi_2 = \langle y, y_1\rangle \dots \langle y_k, z\rangle$. We have that $\mathbf{c}(\Lambda_x(\pi_1)) = \sigma(\lambda_x(\langle x, x_1\rangle))$ and $\mathbf{c}(\Lambda_y(\pi_2)) = \sigma(\lambda_y(\langle y, y_1\rangle))$.

Consider the case $x = y$. By definition of σ, we have that $\sigma(\lambda_x(\langle x, x_1\rangle)) = \sigma(\lambda_y(\langle y, y_1\rangle))$; thus, $\mathbf{c}(\Lambda_x(\pi_1)) = \mathbf{c}(\Lambda_y(\pi_2))$.

Consider the case $x \neq y$. Since, by hypothesis, all the labels used by x are different from all the labels used by y, we have $\sigma(\lambda_x(\langle x, x_1\rangle)) \neq \sigma(\lambda_y(\langle y, y_1\rangle))$ and $\mathbf{c}(\Lambda_x(\pi_1)) \neq \mathbf{c}(\Lambda_y(\pi_2))$.

It is easy to see that the corresponding backward consistent decoding function is

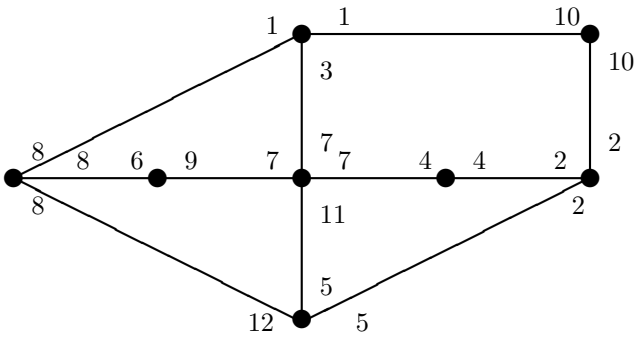


FIG. 17. $\Sigma_x \cap \Sigma_y = \emptyset$.

the following:

$$\forall \pi \in P[x_0], \pi = \langle x_0, x_1 \rangle \dots \langle x_{m-1}, x_m \rangle, \forall \langle x_m, y \rangle \in E(x_m),$$

$$\mathbf{b}(\mathbf{c}(\Lambda_{x_0}(\pi), \lambda_{x_m}(\langle x_m, y \rangle))) = \mathbf{c}(\Lambda_{x_0}(\pi)). \qquad \square$$

Analogously, when the labels used for the edges terminating in x are different from the ones used for edges terminating in any $y \neq x$, then the system has sense of direction. Let $\Sigma_x^- = \{\Sigma_y(y, x) : y \in E(x)\}$.

THEOREM 33. *If, in* (G, λ), $\Sigma_x^- \cap \Sigma_y^- = \emptyset \ \forall \ x \neq y$, *then* (G, λ) *has sense of direction.*

Notice that among the labelings satisfying the assumption of Theorem 32 are those yielding the systems with complete blindness discussed in Theorem 2; among the labelings satisfying the assumption of Theorem 33 are the so-called neighboring ones. In both cases the number of labels used is exactly $n(G)$.

7. Computational equivalence. From a computational viewpoint, *anonymous* systems are the least powerful distributed systems—and hence the ideal setting in which to study the capabilities of specific properties (e.g., [1, 2, 7, 16, 22, 35, 36, 38, 39, 43, 44]). The computational power of sense of direction in anonymous systems has been investigated for specific labeled graphs (e.g., meshes and tori [4, 34], hypercubes [28, 29], and Cayley graphs [27]). The general characterization has been given in [22]; as an indication of its capabilities, many unsolvable problems in anonymous networks (e.g., computing the XOR in a regular network without knowledge of the network size) can be solved if the system has sense of direction (and without breaking anonymity). To exist, and thus to be exploited, sense of direction requires the existence of constraints much stronger than local orientation.

What happens, from a computational point of view, if the system does not even guarantee local orientation? What can be computed if nodes cannot even distinguish among their links? There are several studies on computability in the absence of local orientation investigating these questions for specific topologies [14], specific problems [45], or classes of functions [9, 37]. As expected, these results indicate that the absence of local orientation dramatically reduces the computational power in almost all graphs (a noticeable exception is the ring [14]).

In this section we study the computational capability of the newly introduced backward consistency, which can exist even in the presence of complete and total blindness. We show the unexpected result that *backward consistency has the same computational capabilities as sense of direction*. In other words, not only does it overcome the handicap of not having local orientation, but it also actually empowers even a totally blind system with the additional capabilities of sense of direction.

7.1. Computability. A basic concept when computing on anonymous networks is the one of *view*, introduced in [44]. The *view* $T_{(G,\lambda)}(v)$ of a node v in a labeled graph (G,λ) is an infinite, labeled, rooted tree "downward locally isomorphic" to G, i.e., such that there exists a map from the vertices of the tree to the vertices of G which maps the root of the tree to v, the children of the root to the neighbors of v, and, recursively, the children of a node to the neighbors of that node. The labeling of the arcs are also downward preserved. When no ambiguity arises, we shall denote a view $T_{(G,\lambda)}(v)$ simply by $T(v)$.

Let $\mathcal{G}$ denote the set of labeled graphs. Given $\mathcal{C} \subseteq \mathcal{G}$ and a graph G, let $\mathcal{C}/G$ denote the restriction of $\mathcal{C}$ to G; e.g., $\mathcal{L}^-/G$ is the set of labeled graphs obtained by labeling G with backward local orientations.

A problem $\mathcal{P}$ is *solvable* in $\mathcal{C}/G$ if it is solvable in all labeled graphs $(G,\lambda) \in \mathcal{C}/G$. Given a topological knowledge K, a problem $\mathcal{P}$ is K-solvable in $\mathcal{C}/G$ if it is solvable in all $(G,\lambda) \in \mathcal{C}/G$, where all the nodes are empowered with a priori knowledge K.

We first report two results on the computational capabilities of sense of direction. The first result is that, from a computational point of view, there is no difference between weak sense of direction and sense of direction.

THEOREM 34 (see [22]). *$\forall G \forall \mathcal{P}$, $\mathcal{P}$ is solvable in $\mathcal{W}/G$ iff $\mathcal{P}$ is solvable in $\mathcal{D}/G$.*

The second result shows that, with sense of direction, no other knowledge is necessary.

THEOREM 35 (see [22]). *$\forall G \forall \mathcal{P} \forall K$, if $\mathcal{P}$ is K-solvable in $\mathcal{L}/G$, then $\mathcal{P}$ is solvable in $\mathcal{W}/G$.*

We will now focus on backward consistency and establish the equivalence result.

THEOREM 36. *$\forall G \forall \mathcal{P}$, $\mathcal{P}$ is solvable in $\mathcal{D}/G$ iff $\mathcal{P}$ is solvable in $\mathcal{D}^-/G$.*

Proof. To prove this theorem we will need several steps.

Given two labeled graphs $(G=(V,E),\lambda)$ and $(G'=(V',E'),\lambda')$, a *labeled graph isomorphism* is a bijection $\chi : V \to V'$ which preserves edges and edge labels; that is, $\langle u,v\rangle \in E \Leftrightarrow \langle \chi(u),\chi(v)\rangle \in E'$ and $\lambda(\langle u,v\rangle) = \lambda'(\langle \chi(u),\chi(v)\rangle)$.

The *complete topological awareness* of a node $x \in V$ in a labeled graph $(G = (V,E),\lambda)$ is knowledge of the isomorphic image $\psi(G,\lambda)$ of (G,λ) and of $\phi(x)$ for some labeled graph isomorphism ψ. The *complete topological knowledge* on a labeled graph $(G=(V,E),\lambda)$ (denoted by TK) is when every node $x \in V$ has complete topological awareness. Note that the isomorphism known by each node may be different.

The importance of TK becomes apparent with the following fact.

LEMMA 15 (see [22]). *$\forall G \forall \mathcal{P}$, $\mathcal{P}$ is TK-solvable in $\mathcal{L}/G$ iff $\mathcal{P}$ is solvable in $\mathcal{D}/G$.*

In other words, TK represents the maximum information obtainable with sense of direction. Hence, to prove the main theorem, we need only show that every node can construct TK with weak backward sense of direction. To do this, first observe the following.

LEMMA 16 (see [22]). *Let $\mathbf{c}$ be a consistent coding function of (G,λ). Then $\forall u \in V$, $T_{(G,\lambda)}/\mathbf{c} = \psi(G,\lambda)$ for some labeled graph isomorphism ψ.*

That is, consistency would allow each node to construct an isomorphic image of (G, λ) from each view. Since in its view a node knows its location (it is the root), it knows its location in the image. Therefore, to complete the proof, we have to show how to constructively empower each node with local orientation and a consistent coding.

If there is backward consistency in (G, λ), a local orientation with forward consistency can be easily constructed by employing the result of Lemma 9: if (G, λ) has backward consistency, then $(G, \tilde{\lambda})$ has local orientation and consistency. Since $\tilde{\lambda}$ is distributedly constructible (i.e., each node x can construct $\tilde{\lambda}_x$) by a simple round of communication), $T_{(G,\tilde{\lambda})}(u)$ is also constructible at every node.

Thus, we must only show how to construct at each node a consistent coding function $\mathbf{c}$ for $(G, \tilde{\lambda})$, given $\mathbf{b}$, λ_x, and $\tilde{\lambda}_x$. This is done by observing that, by Theorem 18, if (G, λ) has backward consistent coding $\mathbf{b}$, then we can construct a consistent coding $\mathbf{c}$ of (G, λ^2), and by Lemma 9, $\mathbf{c}$ is a consistent coding for $(G, \tilde{\lambda})$.

Summarizing, if (G, λ) has backward consistency $\mathbf{b}$, each node v can construct the view $T_{(G,\tilde{\lambda})}(v)$ and a consistent coding of $(G, \tilde{\lambda})$ which will empower it to construct both an isomorphic image of both (G, λ) and $\psi(v)$. This knowledge is equivalent to complete topological knowledge; since the class of problems solvable with TK is exactly the same solvable with sense of direction alone, it follows that backward sense of direction is computationally equivalent to sense of direction. □

7.2. Complexity. The result of Theorem 36 opens the problem of how to effectively use the computational power of backward consistency. The approach suggested by the proof of Theorem 36 is to use backward consistency to "simulate" sense of direction. While theoretically valid, this approach sidesteps the problem since it does not exploit backward consistency directly. Furthermore, the technique employed in the proof is not algorithmically reasonable, since it requires the construction of the views $T_{(G,\tilde{\lambda})}(v)$, a task with a formidable communication complexity.

While not solving the general problem, we will provide some complexity relief by pointing out how the simulation approach can be carried out simply and efficiently, without construction of the views.

Let A be an algorithm which solves problem $\mathcal{P}$ in any system with sense of direction. The simulation $S(A)$ of A in a system (G, λ) in which there is backward sense of direction is performed in two stages:

(1) *Preprocessing.* Each processor x computes the set $\Sigma_x(a) = \{b : \lambda_x(x, y) = a$ and $\lambda_y(y, x) = b\}$ for each of its labels $a \in \lambda_x(E(x))$. Notice that $\bigcup \Sigma(a) = \tilde{\lambda}_x(E(x))$.

(2) *Simulation.* Whenever, in algorithm A, a node x sends a message m on an edge labeled l, in the simulation $S(A)$ it will send a message (m, l) to the set of edges labeled by the (unique) label $p \in \lambda_x(E(x))$ such that $l \in \Sigma_x(p)$; whenever, in algorithm A, a node performs an action O upon reception of a message m from a link labeled l, in the simulation of A it will perform the same action O upon reception of message (m, l) from any of the edges labeled p, where p is the (unique) label such that $l \in \Sigma_x(p)$.

It is easy to see that algorithm $S(A)$ behaves on (G, λ) in exactly the same way as the algorithm A would behave on $(G, \tilde{\lambda})$ (the only difference being that $S(A)$ sends a pair $(m, label)$ every time A sends a message m). In fact, we have the following result.

THEOREM 37. *Algorithm $S(A)$ solves $\mathcal{P}$ in any system with backward sense of direction iff A solves $\mathcal{P}$ in any system with sense of direction.*

Proof. Let A solve P in any system with sense of direction. Since (G, λ) has

backward sense of direction, from Theorem 19, we have that $(G, \tilde{\lambda})$ is a system with sense of direction; and thus A solves P on $(G, \tilde{\lambda})$. But, by construction, algorithm S behaves on (G, λ) in exactly the same way as the algorithm A would behave on $(G, \tilde{\lambda})$, and the proof follows.

The reverse is analogous. □

Moreover, the number of message *transmissions* of $S(A)$ (denoted by $\mathcal{M}_T$) is the same as in the original algorithm A. The number of message *receptions* (denoted by $\mathcal{M}_R$) obviously depends on the cardinality of the sets $\Sigma_x(p)$. Given (G, λ), let $h(G, \lambda) = Max_{x \in V, a \in \Sigma}\{\Sigma_x(a)\}$; clearly, $h(G, \lambda) \leq d(G)$.

THEOREM 38.

(1) $\mathcal{M}_T(S(A), G, \lambda) = \mathcal{M}_T(A, G, \tilde{\lambda})$.

(2) $\mathcal{M}_R(S(A), G, \lambda) \leq h(G, \lambda)\mathcal{M}_R(A, G, \tilde{\lambda})$.

In other words, it is possible to use simulation with some level of efficiency. However, the real task is to develop protocols and techniques which exploit backward consistency directly (not just to simulate forward consistency).

8. Concluding remarks and open problems. Sense of direction, for its existence, requires the presence of many "low-level" subconditions, in particular local orientation. However, local orientation cannot be assumed if we are to model systems with more advanced communication and interconnection technology. In this paper we have shown the existence of another type of consistency—backward consistency—which is computationally equivalent to sense of direction but does not require local orientation; thus, it can be found (or designed) in advanced distributed systems.

The relationship between backward consistency and communication complexity is still virtually unknown. The study of how to effectively and efficiently exploit the power of backward consistency when solving problems is an open research area. Some results already exist for specific problems; see, for example, the protocols developed for multi-hop radio networks [3, 12].

Sense of direction and backward consistency are two among a large (and practically unknown) population of varying types of general consistency. An interesting aspect of the results presented here is that backward consistency and sense of direction are not comparable, but rather orthogonal. This fact raises the possibility of the existence of yet other types of consistency that are computationally equivalent to, but not comparable to, sense of direction.

There are interesting coincidences and analogies; for example, the discovery that the reverse of labeling with an important property (sense of direction) has another important property (backward consistency) finds its parallel in the context of interval routing, where it was recently discovered that the reversal of any interval-routing labeling is a broadcast labeling [13]. This coincidence reveals a new viewpoint from which to examine the problem: any operation on the labeling will transform the class of problem solvable in the resulting system (the result will obviously depend on the original type of consistency). The open questions then become, What operations on the labeling (such as reversal) preserve computability? Which will transform it? And into what will it be transformed?

REFERENCES

[1] D. ANGLUIN, *Local and global properties in networks of processors*, in Proceedings of the 12th ACM Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1980, pp. 82–93.

[2] H. Attiya, M. Snir, and M.K. Warmuth, *Computing on an anonymous ring,* J. ACM, 35 (1988), pp. 845–875.
[3] R. Bar-Yehuda, O. Goldreich, and A. Itai, *On the time complexity of broadcast in multi-hop radio networks,* J. Comput. System Sci., 45 (1992), pp. 104–126.
[4] P.W. Beame and H.L. Bodlaender, *Distributed computing on transitive networks: The torus*, in Proceedings of the 6th Symposium on the Theoretical Aspects of Computer Science, Springer-Verlag, New York, 1989, pp. 294–303.
[5] G. V. Bochmann, P. Flocchini, and D. Ramazani, *Distributed objects with sense of direction*, in Proceedings of the IEEE Workshop on Distributed Data and Structures, Orlando, FL, Carleton Scientific, Waterloo, ON, Canada, 1998, pp. 1–15.
[6] H.L. Bodlaender, S. Moran, and M. Warmuth, *The distributed bit complexity of the ring: From the anonymous to the non-anonymous case*, Inform. Comput., 118 (1994), pp. 34–50.
[7] P. Boldi, B. Codenotti, P. Gemmell, S. Shammah, J. Simon, and S. Vigna, *Symmetry breaking in anonymous networks: Characterizations*, in Proceedings of the 4th Israeli Symposium on the Theory of Computing and Systems, Jerusalem, IEEE Press, Piscataway, NJ, 1996, pp. 16–26.
[8] P. Boldi and S. Vigna, *On some constructions which preserve sense of direction*, in Proceedings of the 3rd International Colloquium on Structural Information and Communication Complexity, Siena, Carleton Scientific, Waterloo, ON, Canada, 1996, pp. 47–57.
[9] P. Boldi and S. Vigna, *Computing vector functions on anonymous networks*, in Proceedings of the 4th International Colloquium on Structural Information and Communication Complexity, Ascona, Carleton Scientific, Waterloo, ON, Canada, 1997, pp. 201–204.
[10] P. Boldi and S. Vigna, *Minimal sense of direction and decision problems for Cayley graphs,* Inform. Process. Lett., 64 (1997), pp. 299–303.
[11] P. Boldi and S. Vigna, *Complexity of deciding sense of direction,* SIAM J. Comput., 29 (1999), pp. 779–789.
[12] I. Cidon and O. Mokryn, *Propagation and leader election in a multi-hop broadcast environment*, in Proceedings of the 12th International Symposium on Distributed Computing (DISC 98), Andros, Springer-Verlag, New York, 1998, pp. 104–118.
[13] P. de la Torre, L. Narayanan, and D. Peleg, *The neighbor's interval is greener: A proposal for exploiting interval routing schemes*, in Proceedings of the 5th International Colloquium on Structural Information and Communication Complexity, Amalfi, Carleton Scientific, Waterloo, ON, Canada, 1998, pp. 214–228.
[14] K. Diks, E. Kranakis, A. Malinowski, and A. Pelc, *Anonymous wireless rings,* Theoret. Comput. Sci., 145 (1995), pp. 95–109.
[15] K. Diks, E. Kranakis, and A. Pelc, *Broadcasting in unlabeled tori,* Parallel Process. Lett., 8 (1998), pp. 177–188.
[16] P. Ferragina, A. Monti, and A. Roncato, *Trade-off between computational power and common knowledge in anonymous rings*, in Proceedings of the 1st International Colloquium on Structural Information and Communication Complexity, Ottawa, Carleton Scientific, Waterloo, ON, Canada, 1994, pp. 35–48.
[17] P. Flocchini, *Minimal sense of direction in regular networks,* Inform. Process. Lett., 61 (1997), pp. 331–338.
[18] P. Flocchini and B. Mans, *Optimal election in labeled hypercubes,* J. Parallel Distrib. Comput., 33 (1996), pp. 76–83.
[19] P. Flocchini, B. Mans, and N. Santoro, *On the impact of sense of direction on message complexity,* Inform. Process. Lett., 63 (1997), pp. 23–31.
[20] P. Flocchini, B. Mans, and N. Santoro, *Sense of direction: Definition, properties and classes,* Networks, 32 (1998), pp. 165–180.
[21] P. Flocchini, B. Mans, and N. Santoro, *Sense of direction in distributed computing*, Theoret. Comput. Sci., to appear.
[22] P. Flocchini, A. Roncato, and N. Santoro, *Computing on anonymous networks with sense of direction,* Theoret. Comput. Sci., to appear.
[23] P. Flocchini, A. Roncato, and N. Santoro, *Symmetries and sense of direction in labeled graphs,* Discrete Appl. Math., 87 (1998), pp. 99–115.
[24] P. Flocchini and N. Santoro, *Topological constraints for sense of direction,* Internat. J. Found. Comput. Sci., 9 (1998), pp. 179–198.
[25] C. Gavoille, *A survey on interval routing,* Theoret. Comput. Sci., 245 (2000), pp. 217–253.
[26] Z. Harpaz and S. Zaks, *Sense of Direction in Complete Distributed Networks,* Tech. Report TR-453, Department of Computer Science, Technion, Haifa, Israel, 1987. Abstract in Proceedings of the 23rd Allerton Conference on Communication, Control and Computing, 1985.

[27] E. Kranakis and D. Krizanc, *Labeled versus unlabeled distributed Cayley networks,* Discrete Appl. Math., 63 (1995), pp. 223–236.

[28] E. Kranakis and D. Krizanc, *Distributed computing on anonymous hypercubes,* J. Algorithms, 23 (1997), pp. 32–50.

[29] E. Kranakis and N. Santoro, *Distributed computing on anonymous hypercubes with faulty components,* Distrib. Comput., 14 (2001), pp. 185–189.

[30] M.C. Loui, T.A. Matsushita, and D.B. West, *Election in complete networks with a sense of direction,* Inform. Process. Lett., 22 (1986), pp. 185–187. See also the corrigendum in Inform. Process. Lett., 28 (1988), p. 327.

[31] B. Mans, *Optimal distributed algorithms in unlabeled tori and chordal rings,* J. Parallel Distrib. Comput., 46 (1997), pp. 80–90.

[32] B. Mans and N. Santoro, *On the impact of sense of direction in arbitrary networks*, in Proceedings of the 14th International Conference on Distributed Computing Systems, Poznan, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 258–265.

[33] T. Masuzawa, N. Nishikawa, K. Hagihara, and N. Tokura, *Optimal fault-tolerant distributed algorithms for election in complete networks with a global sense of direction*, in Proceedings of the 3rd International Workshop on Distributed Algorithms, Springer-Verlag, New York, 1989, pp. 171–182.

[34] A. Monti and A. Roncato, *A gap theorem for anonymous torus,* Inform. Process. Lett., 57 (1996), pp. 279–285.

[35] S. Moran and M.K. Warmuth, *Gap theorems for distributed computation,* SIAM J. Comput., 22 (1993), pp. 379–394.

[36] N. Norris, *Classifying anonymous networks: When can two networks compute the same vector-valued functions?*, in Proceedings of the 1st International Colloquium on Structural Information and Communication Complexity, Ottawa, Carleton Scientific, Waterloo, ON, Canada, 1994, pp. 83–98.

[37] N. Norris, *Computing functions on partially wireless networks*, in Proceedings of the 2nd International Colloquium on Structural Information and Communication Complexity, Olympia, Carleton Scientific, Waterloo, ON, Canada, 1995, pp. 53–64.

[38] N. Norris, *Universal covers of graphs: Isomorphism to depth $n-1$ implies isomorphism to all depths,* Discrete Appl. Math., 56 (1995), pp. 61–74.

[39] A. Roncato, *Gap theorems for anonymous rings*, in Proceedings of the 2nd International Colloquium on Structural Information and Communication Complexity, Olympia, Carleton Scientific, Waterloo, ON, Canada, 1995, pp. 65–76.

[40] N. Santoro, J. Urrutia, and S. Zaks, *Sense of direction and communication complexity in distributed networks*, in Proceedings of the 1st International Workshop on Distributed Algorithms, Ottawa, Carleton Univeristy Press, Ottawa, ON, Canada, 1985, pp. 123–132.

[41] G. Singh, *Efficient leader election using sense of direction*, Distrib. Comput., 10 (1997), pp. 159–165.

[42] G. Tel, *Sense of direction in processor networks*, in Proceedings of the Conference on Theory and Practice of Informatics, Springer-Verlag, New York, 1995, pp. 50–82.

[43] M. Yamashita and T. Kameda, *Computing functions on asynchronous anonymous networks,* Math. Systems Theory, 29 (1996), pp. 331–356.

[44] M. Yamashita and T. Kameda, *Computing on anonymous networks, part* I: *Characterizing the solvable cases*, IEEE Trans. Parallel Distrib. Comput., 7 (1996), pp. 69–89.

[45] M. Yamashita and T. Kameda, *Leader election problem on networks in which processor identity numbers are not distinct*, IEEE Trans. Parallel Distrib. Systems, 10 (1999), pp. 878–887.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 307–325

BINARY SPACE PARTITIONS FOR LINE SEGMENTS WITH A LIMITED NUMBER OF DIRECTIONS*

CSABA D. TÓTH†

Abstract. We show that there is always a binary space partition (BSP) of size $O(n \log k)$ and an autopartition of size $O(nk)$ for n disjoint line segments in the plane, assuming that the segments have k distinct orientations. In particular, if k is a constant, these bounds imply that there is a linear-size BSP and autopartition. Our proof is constructive and can be turned into algorithms computing such a BSP or autopartition in $O(n^2)$ and $O(n^2 k)$ times.

Key words. binary space partition, line segments, computational geometry

AMS subject classifications. 52C45, 68P05, 68Q25, 68U05

PII. S0097539702403785

1. Introduction.

The binary space partition (BSP) is a data structure introduced by the computer graphics community in the late seventies [17, 24, 26]. Originally it was designed to give an efficient visibility algorithm for polygonal scenes using the "painter's method." Currently applications of BSP range through hidden surface removal [18, 13], shadow generation [6], ray tracing [8, 25], solid modeling [19], surface approximation [20, 28], and robot motion planning [7, 4].

Informally, a BSP for n disjoint objects in the plane is a recursive dissection of the plane into convex regions such that each region is dissected in two by a segment or a line until every object (or part of an object) is in a distinct region. Ideally, every object should be in one convex region, but sometimes it is inevitable that some of the objects are dissected. The size of the BSP is defined as the number of regions in the resulting plane partition. The size of a BSP effects substantially the complexity of the algorithms using this data structure. Our concern, in general, is to find small BSPs for certain sets of objects. A special but important class of BSPs consists of *autopartitions* where every convex region is dissected along a flat of one of the objects within the region.

Previous and related works. The BSP was invented by Fuchs, Kedem, and Naylor [17], based on ideas of Schumacker et al. [24, 26]. Paterson and Yao [22] proved that there exists a BSP of size $O(n \log n)$ for a set of n disjoint line segments in the plane. The expected size of an autopartition, where cuts are made along the line segments in a random order, is $O(n \log n)$. The best-known lower bound, $\Omega(n \log n / \log \log n)$, for both BSP and autopartition follows from a recent construction described in [27]. De Berg, de Groot, and Overmars [11] have found linear upper bounds to the BSP for line segments where the ratio between the length of the longest and shortest segment is bounded by a constant.

*Received by the editors March 11, 2002; accepted for publication September 18, 2002; published electronically January 17, 2003. Part of this paper was published in *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, CA, SIAM, Philadelphia, 2002. This work was supported by the joint Berlin-Zürich graduate program "Combinatorics, Geometry, and Computation," which was financed by the German Science Foundation (DFG) and ETH Zürich.

http://www.siam.org/journals/sicomp/32-2/40378.html

†Institute for Theoretical Computer Science, ETH Zürich, CH-8092, Zürich, Switzerland (toth@inf.ethz.ch).

Paterson and Yao [23, 21] exhibited BSPs of size $O(n^2)$ for rectangles in $\mathbb{R}^3$, matching the $\Omega(n^2)$ lower bound of a construction due to Eppstein. Agarwal et al. [2] constructed a BSP of size $n2^{O(\sqrt{\log n})}$ for n fat orthogonal rectangles in $\mathbb{R}^3$ in $n2^{O(\sqrt{\log n})}$ time. Dumitrescu, Mitchell, and Sharir [14] have proved that the size of the smallest BSP for n axis-parallel 2-dimensional rectangles in $\mathbb{R}^4$ is $\Theta(n^{5/3})$ in the worst case. De Berg [10] gave BSPs of size $O(n)$ for n "uncluttered" objects in $\mathbb{R}^d$ for every $d \in \mathbb{N}$ in $O(n \log n)$ time.

Main result. The construction of [27] shows that, in general, we cannot expect linear-size BSPs for sets of line segments in the plane. We may ask what is the smallest BSP if n segments have few different orientations compared to n, which is the case in many applications. Paterson and Yao [22] proved an $O(n)$ upper bound on the smallest BSP for orthogonal line segments. Dumitrescu, Mitchell, and Sharir [14] improved this upper bound to $2n-1$ and gave a worst-case lower bound of $2n-o(n)$. For more than two distinct directions (even for three different orientations) no better bound was known than $O(n \log n)$. We have the following.

THEOREM 1. *If L is a set of n disjoint line segments in the plane with at most k distinct directions, then there exists for L*

(i) *a BSP of size $O(n \log k)$, where every segment is cut at most $O(\log k)$ times;*

(ii) *an autopartition of size $O(nk)$, where every segment is cut at most $O(k)$ times.*

If k is a constant, Theorem 1 yields a linear upper bound, which is, up to a constant factor, optimal. Interestingly enough, the bound of Theorem 1(i) is also asymptotically tight for the lower bound construction of [27]: Here n line segments have $\exp(\Theta(\log n/\log\log n))$ different directions, and the minimum-size BSP is as big as $\Omega(n \log n/\log\log n) = \Omega(n \log k)$. For $k = n$, Theorem 1(i) returns the well-known bound of $O(n \log n)$ due to Paterson and Yao [22].

Our proof is constructive. The algorithm of constructing a BSP of size $O(n \log k)$ and an autopartition of size $O(nk)$ uses $O(n^2)$ and $O(n^2k)$ times, respectively. For $k = 2$, d'Amore and Franciosa [9] gave an $O(n \log n)$ time algorithm to find an autopartition of size at most $4n$.

Definitions. We define the BSP and the autopartition in the Euclidean plane only. It can be defined analogously in higher-dimensional Euclidean spaces (see, e.g., [12]). A *binary space partition tree* P for a set L of n disjoint line segments in the plane is a *rooted binary tree* defined as follows. Each node $u \in P$ corresponds to a convex region R_u. The root of P corresponds to $\mathbb{R}^2$ (or, equivalently, to the convex hull of all segments). If $\operatorname{int}(R_u)$ is disjoint from the segments, then u is a leaf of P. Otherwise R_u is partitioned into two convex regions R_v and R_w along a line, and the two children v and w correspond to R_v and R_w, respectively. A BSP is an *autopartition* if each R_u, where the interior of R_u intersects a segment of L, is dissected along a line spanned by one of the segments intersecting $\operatorname{int}(R_u)$.

The size of a BSP P is the number of its leaves. We denote by $\mathcal{R}(P)$ the set of regions corresponding to the leaves of P. The regions in $\mathcal{R}(P)$ give a convex decomposition of the plane. Note that many different BSPs and autopartitions exist for the same set L. The size of a BSP depends largely on the choice of the line that splits R_u at each interior node u. In fact, we consider a BSP for L as a recursive *algorithm* of building the tree from the root, where we choose a splitting line whenever there is a node u for which $\operatorname{int}(R_u)$ intersects a segment and split R_u into two regions R_v and R_w.

Proof technique and organization. Instead of the size of the BSP, we count

the number $c(P)$ of *cuts* on line segments during a BSP P, i.e., the events where a splitting line of an interior node $v \in P$ crosses a segment $\ell \in L$. The number of pieces of line segments is thus $n + c(P)$. The size of every BSP P described below is bounded by $O(n + c(P))$, since all our BSPs predominantly partition regions along fragments of the given line segments, and every fragment must be contained in one of the partitioning lines. Therefore, it is enough to show that there is a BSP (resp., autopartition) P with $c(P) = O(n \log k)$ (resp., $O(nk)$) for n disjoint line segments with k distinct directions.

We construct BSPs and autopartitions recursively. In the main loop, we apply a BSP for the segments which have an endpoint on the boundary of the convex hull $\mathrm{conv}(\bigcup L)$. Our Lemma 1 ensures that such a BSP *cut*s every line segment at most $O(\log k)$ and $O(k)$ times in the case of BSPs and autopartitions, respectively. Every fragment of segments cut during one iteration will have an endpoint on the boundary of a corresponding cell R_v, and we apply a BSP for those segments in the next iteration. In section 2, we describe the main loop of our algorithm and the proof of Theorem 1.

The proof of Lemma 1 can be found in sections 5 and 6. The two proofs (for the cases of BSP and autopartition) use recursion, too, and rely on simple partitioning algorithms on *convex sequences* and *cycles*. We discuss these concepts in detail in section 3.

2. Main theorem. First we introduce some notation. Fix a set L of disjoint line segments in the plane. We denote by $\mathrm{int}(C)$ and ∂C the interior and the boundary, respectively, of a planar set C. A *cell* is defined as a bounded closed convex region in the plane. We avoid considering unbounded convex regions by associating the root of every BSP to the convex hull $\mathrm{conv}(\bigcup L)$ of segments (instead of $\mathbb{R}^2$). Consequently, every node u of a BSP corresponds to a cell R_u.

For a cell C, let $L(C) = \{\ell \cap \mathrm{int}(C) : \ell \in L\} \setminus \{\emptyset\}$, that is, the set of fragments of line segments from L within the cell C. We use a simple observation throughout this paper: If there is a segment $\ell \in L(R_u)$ for some interior node u of the BSP such that ℓ connects two points on the boundary of the cell R_u, then we can split R_u along ℓ without creating any new cuts on segments. We apply such *free cuts* whenever possible. If a segment $\ell \in L$ is dissected into many pieces by consecutive cuts, we only have to care about two pieces, the two *endings* of ℓ, which are two disjoint line segments in two nonoverlapping regions of the current convex partition.

We may assume that no segment of $L(C)$ has two endpoints on the boundary of C, since otherwise a free cut could split C into two cells. We partition $L(C)$ into two disjoint subsets: Let $B(C)$ be the set of *boundary segments*, i.e., segments from $L(C)$ with exactly one endpoint on the boundary of C (and the other endpoint in $\mathrm{int}(C)$). Let $I(C)$ be the set of *interior segments*, i.e., segments from $L(C)$ which lie entirely in the interior of C.

Suppose without loss of generality (w.l.o.g.) that none of the segments of L is parallel to the x-axis of our coordinate system. Let $B^+(C) \subset B(C)$ (resp., $B^-(C) \subset B(C)$) be the set of all segments $\ell \in B(C)$ such that the extension of ℓ beyond the boundary ∂C is y-monotone increasing (resp., decreasing). Since the roles of $B^+(C)$ and $B^-(C)$ are symmetric, the claims we formulate below for $B^+(C)$ hold equally for $B^-(C)$, too.

The following lemma is the key to the proof of Theorem 1(i) and 1(ii). The proof of this lemma is postponed until the last two sections.

LEMMA 1. *Assume that we are given a nonempty set L of disjoint line segments in the plane with at most k distinct directions and a cell C.*

(i) *There exists a BSP P_C for $B^+(C)$ where every line segment of $B(C)$ is cut at most $3\lfloor \log_2 k \rfloor + 8$ times and every line segment of $I(C)$ is cut at most $6\lfloor \log_2 k \rfloor + 11$ times.*

(ii) *There exists a BSP P_C for $B^+(C)$ where every line segment of $B(C)$ is cut at most $4k + 4$ times and every line segment of $I(C)$ is cut at most $4k + 7$ times. Moreover, every region R_u corresponding to an internal node $u \in P_C$ is partitioned along a segment of $L(R_u)$.*

The last sentence of Lemma 1(ii) does not say that the BSP P_C is an autopartition for $B^+(C)$: Possibly P_C uses splitting lines spanned by segments of $L(C)$ which are actually not in $B^+(C)$. This constraint, however, will be used below to establish the existence of an autopartition for $L(C)$. A BSP P for $L(C)$ is now constructed by the following recursive algorithm.

Algorithm 1.

- *Put $i := 1$ and $\mathcal{R}_1 := \{\text{conv}(\bigcup L)\}$.*
- *While there is a $C \in \mathcal{R}_i$ such that $L(C)$ is nonempty, do*
  1. *If i is odd, then for every cell $C \in \mathcal{R}_i$ where $B^+(C)$ is nonempty, partition C applying Lemma 1 to $B^+(C)$.*
  2. *If i is even, then for every cell $C \in \mathcal{R}_i$ where $B^-(C)$ is nonempty, partition C applying* Lemma 1 *to $B^-(C)$.*
  3. *In every cell $C \in \mathcal{R}_i$ where $B(C)$ is empty but there is a segment $\ell \in I(C)$, dissect C along ℓ.*
  4. *Apply all possible free cuts in the resulting cells.*
  5. *Let $\mathcal{R}_{i+1}$ be the set of all resulting cells, and put $i := i + 1$.*

Proof of Theorem 1. Algorithm 1 performs a BSP for L. If every cell is dissected along a segment in the cell, then this BSP is an autopartition. From the point of view of one line segment $\ell \in L$, the algorithm appears as follows: ℓ is cut into pieces at some step i_ℓ. (There is the possibility of cutting along ℓ before it is cut otherwise. In this case ℓ is not cut at all by the procedure.) In step i_ℓ, ℓ may be cut several times, and its two endings fall into $B^+(D_1)$ and $B^-(D_2)$, where D_1 and D_2 are distinct cells of $\mathcal{R}_{i_\ell+1}$. Assume w.l.o.g. that i_ℓ is odd. In the even step $i_\ell + 1$, BSPs for $B^-(D_1)$ and $B^-(D_2)$ are called. In this step, both endings may also be cut several times. D_1 may be subdivided and the ending of the portion of ℓ within $B^+(D_1)$ belongs to some $B^+(E_1)$ for a $E_1 \subseteq D_1$, $E_1 \in \mathcal{R}_{i_\ell+2}$. In the odd step $i_\ell + 2$, a BSP for $B^+(E_1)$ is called, but ℓ may still be cut several times. Analogously, if i_ℓ is even, both endings may be cut at step $i_\ell + 1$, but only one of them can be cut in step $i_\ell + 2$. Thus, ℓ may be cut in at most three consecutive steps of Algorithm 1.

For part (i) of Theorem 1, we determine the maximal number of cuts during the "life cycle" of a segment ℓ from Lemma 1(i). In step i_ℓ, segment ℓ is cut at most $6\lfloor \log_2 k \rfloor + 11$ times. In step $i_\ell + 1$, either ending of the segment is cut at most $3\lfloor \log_2 k \rfloor + 8$ times. Finally in step $i_\ell + 2$, its one surviving ending might be cut at most $3\lfloor \log_2 k \rfloor + 8$ times. All in all, ℓ is cut at most $15\lfloor \log_2 k \rfloor + 35$ times during the algorithm.

For the proof of Theorem 1(ii), we use Lemma 1(ii). In step i_ℓ, segment ℓ is cut at most $4k + 7$ times. In step $i_\ell + 1$, either ending of the segment is cut at most $4k + 4$ times. Finally in step $i_\ell + 2$, its one surviving ending can be cut at most $4k + 4$ times. All in all, ℓ is cut at most $(4k + 7) + 3(4k + 4) = 16k + 19$ times during Algorithm 1. □

The constants in the upper bounds on the total number of cuts per segment can be slightly improved by a more careful analysis, namely, by applying the detailed

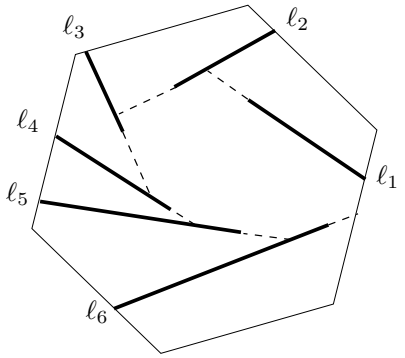


FIG. 1. *$P(\mathcal{S})$ for a convex sequence $\mathcal{S}$.*

versions of Lemma 1(i) and 1(ii) as they appear in sections 5 and 6.

3. Sequences, cycles, and convex cycles. In this section, we define a few simple BSP schemes using simple structures formed by boundary segments. These partition algorithms will serve as building blocks in the proofs of Lemma 1(i) and 1(ii) in the last two sections.

Let C be a cell and let L be a set of segments as above. For this section, consider a fixed set X, $X \subset B(C)$ of boundary segments. Recall that every segment $\ell \in X$ has one endpoint in $\mathrm{int}(C)$. Denote the other endpoint, $\ell \cap \partial C$, by $p(\ell)$. Extend ℓ towards the interior of C until it hits another segment of X or the boundary ∂C and denote the endpoint of the extension by $g(\ell)$. Notice that the extension of ℓ may cross segments of $L(C) \setminus X$ before hitting some other segment of X or ∂C; in other words, the function g depends on $X \subset B(C)$.

3.1. Sequences and cuts along sequences. We define a sequence as follows.

DEFINITION 1. *A* sequence *of X is a t-tuple $(\ell_1, \ell_2, \ldots, \ell_t)$, $t \in \mathbb{N}$, such that*
- *$\ell_1, \ell_2, \ldots, \ell_t \in X$,*
- *$g(\ell_i) \in \ell_{i+1}$ for $i = 1, 2, \ldots, t-1$, and*
- *$g(\ell_t) \in \partial C$.*

DEFINITION 2. *A sequence $(\ell_1, \ell_2, \ldots, \ell_t)$ is* convex *if the point $p(\ell_1)$ together with the points $g(\ell_i)$, $i = 1, 2, \ldots, t$, form a convex $(t+1)$-gon (see Figure 1).*

Consider a convex sequence $\mathcal{S} = (\ell_1, \ell_2, \ell_3, \ldots, \ell_t)$ in X. We obtain a BSP by recursively cutting along the segments of the sequence in reverse order starting from ℓ_t and proceeding to ℓ_1.

ALGORITHM 2 ($P(\mathcal{S})$).
- *Input: cell C, convex sequence $\mathcal{S} = (\ell_1, \ell_2, \ell_3, \ldots, \ell_t)$.*
- *For $j = 0, 1, 2, \ldots, t-1$, do*
  Partition every region D, $D \cap \ell_{t-j} \neq \emptyset$, along the line through ℓ_{t-j}.

PROPOSITION 2. *In $P(\mathcal{S})$, no segment of X is cut, and every segment of $L(C) \setminus X$ is cut at most twice.*

Proof. Since the segments of L are disjoint, the only portion of the partitioning lines $p(\ell_i)g(\ell_i)$, $i = t, t-1, \ldots, 1$, that may cut other segments is the extension of ℓ_i to $g(\ell_i)$. All these portions lie along sides of the convex polygon $p(\ell_1)g(\ell_1)g(\ell_2)\ldots g(\ell_t)$. A convex polygon crosses every line segment at most twice. ∎

Let $\mathcal{T}(P(\mathcal{S}))$ be the union of all the segments along which $P(\mathcal{S})$ made a dissection (i.e., the union of all extended segments $p(\ell_i)g(\ell_i)$, $i = 1, 2, \ldots, t$). Note that $\mathcal{T}(P(\mathcal{S}))$

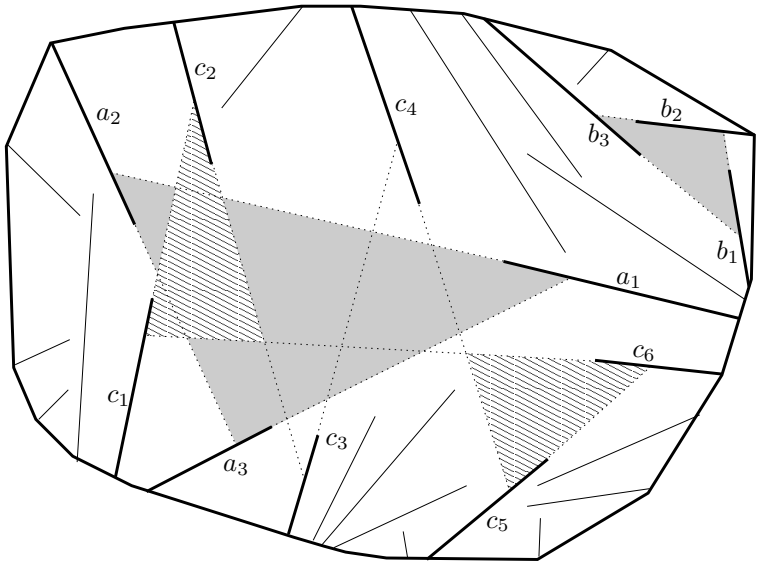


FIG. 2. *Three cycles* (a_1, a_2, a_3), (b_1, b_2, b_3), *and* $(c_1, c_2, c_3, c_4, c_5, c_6)$ *give rise to four convex cycles* (a_1, a_2, a_3), (b_1, b_2, b_3), (c_4, c_5, c_6), *and* (c_6, c_1, c_2). *Segments of cycles are depicted by bold lines. Segments of* $L(C) \setminus X$ *are not indicated.*

is a connected set.

3.2. Cycles and convex cycles.

DEFINITION 3. *A* cycle *of* X *is a* t*-tuple* $(\ell_1, \ell_2, \dots, \ell_t)$, $t \in \mathbb{N}$, *such that*

- $\ell_1, \ell_2, \dots, \ell_t \in X$,
- $g(\ell_i) \in \ell_{i+1}$ *for* $i = 1, 2, \dots, t-1$, *and*
- $g(\ell_t) \in \ell_1$.

Figure 2 exhibits three cycles. Every segment of X may appear in at most one cycle. The concept of *convex cycle*, defined below, is not a subclass of cycles: The extension of one segment in a convex cycle does not necessarily hit the next element of the cycle, but all segments are arranged along the sides of a convex polygon.

DEFINITION 4. *A* convex cycle *of* X *is a* t*-tuple* $(\ell_1, \ell_2, \dots, \ell_t)$ *such that*

- $\ell_1, \ell_2, \dots, \ell_t \in X$,
- $g(\ell_i) \in \ell_{i+1}$ *for* $i = 1, 2, \dots, t-1$,
- *the extension of* ℓ_t *hits the line through* ℓ_1 *at a point* $a \in C$ *but does not cross any segment of* X *until it reaches* a, *and*
- $g(\ell_1)g(\ell_2)\dots g(\ell_{t-1})a$ *forms a convex* t*-gon.*

In Figure 2, the cycles (a_1, a_2, a_3) and (b_1, b_2, b_3) are convex cycles. The third cycle $(c_1, c_2, c_3, c_4, c_5, c_6)$ is not a convex cycle but contains two convex cycles, (c_4, c_5, c_6) and (c_6, c_1, c_2), which are indicated by their striped interior convex regions.

PROPOSITION 3. *For any cycle* $(\ell_1, \ell_2, \dots, \ell_t)$, *there is* $q \in [1, t-2]$ *and* $r \in [q+2, t]$ *such that* $(\ell_q, \ell_{q+1}, \dots, \ell_r)$ *is a convex cycle.*

Proof. Consider the polygonal curve $\gamma = (p(\ell_1), g(\ell_1), g(\ell_2), \dots, g(\ell_t))$. If γ has no self-crossing, then $\mathcal{S}$ is a convex cycle.

Let $(p(\ell_1), g(\ell_1), g(\ell_2), \dots, g(\ell_{r-1}), w) \subset \gamma$ be the longest prefix curve without self-crossing. That is, w denotes the common point of two segments, say, $g(\ell_q)g(\ell_{q+1})$ and $g(\ell_{r-1})g(\ell_r)$, $q+1 < r-1$. We show that $(\ell_q, \ell_{q+1}, \dots, \ell_r)$ is a convex cycle.

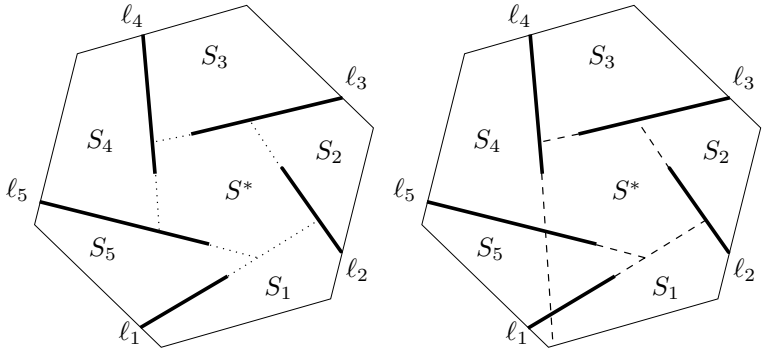


FIG. 3. *The notation on a convex cycle $\mathcal{S}$ (left) and $P^0(\mathcal{S}, 4)$ (right).*

$g(\ell_i) \in \ell_{i+1}$ for $i = q, q+1, \ldots, r-1$; the extension of ℓ_r hits the line through ℓ_q at point w; but it does not hit any segment of X before. Extend every segment ℓ_i for $i = 1, 2, \ldots, r-1$ to $g(\ell_i)$, and ℓ_r to w. We obtain a convex partition of C in which one face is $(g(\ell_q), g(\ell_{q+1}), \ldots, g(\ell_{r-1}), w)$; therefore it is a convex polygon. □

We observe here a useful property of set $B^+(C)$ with respect to cycles.

PROPOSITION 4. *$B^+(C)$ contains no cycle.*

Proof. Suppose, to the contrary, that $(\ell_1, \ell_2, \ell_3, \ldots, \ell_t)$ is a convex cycle and, furthermore, $\ell_i \in B^+(C)$ for every $i = 1, 2, \ldots, t$. Let w be the vertex of the polygon $g(\ell_1)g(\ell_2)\ldots g(\ell_{t-1})a$ with the largest y-coordinate. Point w is the intersection of two lines through, say, ℓ_i and ℓ_{i+1}. Either the extension of ℓ_i or that of ℓ_{i+1} beyond the boundary ∂C is not y-monotone increasing, a contradiction. □

Consider a convex cycle $\mathcal{S} = (\ell_1, \ell_2, \ell_3, \ldots, \ell_t)$ of X. Extend every ℓ_i for $i = 1, 2, \ldots, t-1$ to $g(\ell_i)$, and ℓ_t to a. The extended segments of S partition the cell C into $t+1$ convex regions $S^*, S_1, \ldots, S_t$. One of them, $S^* = g(\ell_1)g(\ell_2)\ldots g(\ell_{t-1})a$, lies completely in the interior of C (Figure 3, left).[1] Clearly, $\mathrm{int}(S^*)$ is disjoint from all segments of X. Every S_i, $i = 1, \ldots, t$, has a common boundary with C. We may choose the notation of S_i, $i = 1, 2, \ldots, t$, such that S_i is bounded by portions of the extension of segments ℓ_i and ℓ_{i+1}, and by ∂C.

3.3. Cuts along convex cycles. Consider a convex cycle $\mathcal{S} = (\ell_1, \ell_2, \ell_3, \ldots, \ell_t)$. A BSP $P^0(\mathcal{S}, i)$, $i = 1, \ldots, t$, can be obtained in the following way (see Figure 3, right).

ALGORITHM 3 ($P^0(\mathcal{S}, i)$).

- *Input: cell C, index i, convex cycle $\mathcal{S} = (\ell_1, \ell_2, \ell_3, \ldots, \ell_t)$.*
- *For $j = 0, 1, 2, \ldots, t-1$, do*
  Partition every region D, $D \cap \ell_{i-j} \neq \emptyset$, along the line through ℓ_{i-j}.

Denote by $\pi(\mathcal{S}, i)$ the set of regions S_i, $i = 1, 2, \ldots, t$, pierced by the line through $\ell_i \in \mathcal{S}$, and let $\bar{\pi}(\mathcal{S}, i) = \{S_1, S_2, \ldots, S_t\} \setminus \pi(\mathcal{S}, i)$. For instance, in Figure 3, $\pi(\mathcal{S}, 4) = \{S_1, S_5\}$.

Remark 1. For any convex cycle $\mathcal{S} = (\ell_1, \ell_2, \ell_3, \ldots, \ell_t)$ and any index i, $1 \leq i \leq t$, we have $S_{i-1} \in \bar{\pi}(\mathcal{S}, i)$ and $S_i \in \bar{\pi}(\mathcal{S}, i)$.

PROPOSITION 5. *The regions of $\bar{\pi}(\mathcal{S}, i)$ are elements of the convex subdivision $\mathcal{R}(P^0(\mathcal{S}, i))$.*

[1] Here and in what follows, arithmetic on the indices of $\ell_1, \ell_2, \ldots, \ell_t$ and $S_1, S_2, \ldots, S_t$ is meant mod t, that is, $t+1$ stands for 1.

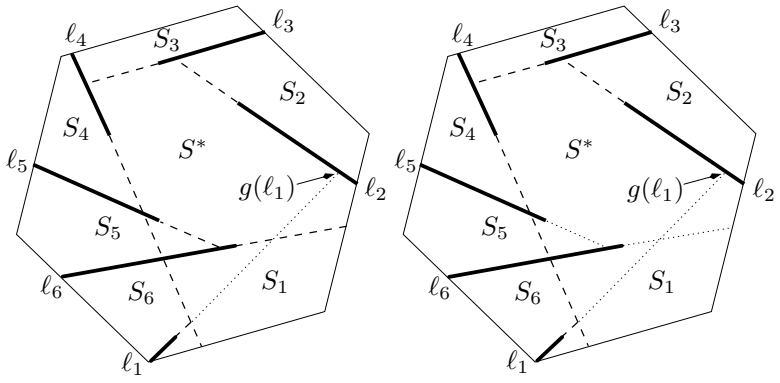


FIG. 4. $P^0(\mathcal{S}, 4)$ *(left) and* $P^1(\mathcal{S}, 4)$ *(right). Dissection lines are dashed.*

Proof. Every region S_j, $j = 1, 2, \ldots, t$, is determined by the cuts along ℓ_{j+1} and along ℓ_j to $g(\ell_j)$. Every segment ℓ_j of $\mathcal{S}$, except for ℓ_i, is extended to $g(\ell_j)$ only. If $S_j \in \bar{\pi}(\mathcal{S}, i)$, then the partitioning lines cut S_j out along its boundaries. □

PROPOSITION 6. *In the BSP* $P^0(\mathcal{S}, i)$*,*

- *every segment of the convex cycle* $\mathcal{S}$ *is cut at most once,*
- *no segment of* X *in a region of* $\pi(\mathcal{S}, i)$ *is cut,*
- *every segment of* X *in a region of* $\bar{\pi}(\mathcal{S}, i)$ *is cut at most twice, and*
- *every segment of* $L(C) \setminus X$ *is cut at most four times.*

Proof. A segment in the cycle $\mathcal{S}$ can only be cut by the line through ℓ_i. By Proposition 5, a segment lying in a region of $\pi(\mathcal{S}, i)$ cannot be cut. A segment entirely in a region of $\bar{\pi}(\mathcal{S}, i)$ can be cut by the line through ℓ_i. It can also be cut by the line through ℓ_t: If the line through ℓ_i separates ℓ_1 and $g(\ell_1)$, then the dissection along ℓ_t beyond the point a may cut again into the region S_1 (see, e.g., Figure 4, left). A segment $\ell \in L(C) \setminus X$ may be cut by the lines through ℓ_i and ℓ_t and by other dissecting lines located along the convex polygon S^*. □

In Algorithm 6 in section 6, we wish to have at most one cut on segments of X within the regions of $\bar{\pi}(\mathcal{S}, i)$; therefore, we need a slightly modified version of $P^0(\mathcal{S}, i)$, as follows.

ALGORITHM 4 ($P^1(\mathcal{S}, i)$).

- *Input: cell* C*, index* i*, convex cycle* $\mathcal{S} = (\ell_1, \ell_2, \ell_3, \ldots, \ell_t)$ *such that the line through* ℓ_i *does not cross the segment* ℓ_1 *but crosses the extended segment* $p(\ell_1)g(\ell_1)$*.*
- *For* $j = 0, 1, 2, \ldots, i-1$*, do*
  Partition every region D*,* $D \cap \ell_{i-j} \neq \emptyset$*, along the line through* ℓ_{i-j}*.*

The line through ℓ_i intersects the segments $\ell_{i+1}, \ell_{i+2}, \ldots, \ell_t$, because $p(\ell_{i+1})$, $p(\ell_{i+2}), \ldots, p(\ell_t)$, and $p(\ell_1)$ are all on the same side of ℓ_i. Therefore the portions of these segments between ∂C and the line through ℓ_i give rise to free cuts once the BSP $P^1(\mathcal{S}, i)$ is completed.

The two main differences between $P^0(\mathcal{S}, i)$ and $P^1(\mathcal{S}, i)$ are that (1) $P^0(\mathcal{S}, i)$ dissects along all segments of S while $P^1(\mathcal{S}, i)$ does not dissect along $\ell_{i+1}, \ell_{i+2}, \ldots, \ell_t$; (2) $P^0(\mathcal{S}, i)$ cuts segments within a region of $\bar{\pi}(\mathcal{S}, i)$ at most twice, but $P^1(\mathcal{S}, i)$ cuts them at most once.

PROPOSITION 7. *In the BSP* $P^1(\mathcal{S}, i)$*,*

- *every segment of the convex cycle* $\mathcal{S}$ *is cut at most once,*

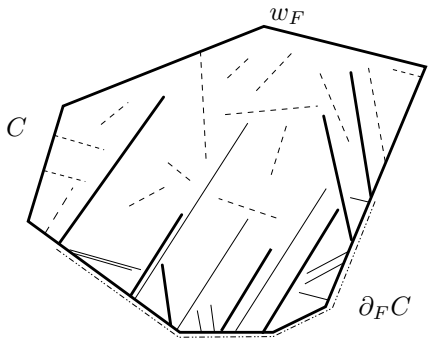


FIG. 5. *Segments of F are bold, segments of F^* are thin, and segments of $L(C) \setminus F^*$ are dashed.*

- *no segment of X in a region of $\pi(\mathcal{S}, i)$ is cut,*
- *every segment of X in a region of $\bar{\pi}(\mathcal{S}, i)$ is cut at most once, and*
- *every segment of $L(C) \setminus X$ is cut at most three times.*

4. More preparations. In sections 5 and 6, we describe two binary space partitioning algorithms for $B^+(C)$ to prove Lemma 1(i) and 1(ii). In intermediate steps of the partitionings in a subcell $D \subset C$, the set $B^+(D)$ may contain strictly more elements than $F = \{\ell \cap D : \ell \in B^+(C)\} \setminus \{\emptyset\}$ (i.e., the set of endings of segments of $B^+(C)$ in D): Although $F \subseteq B^+(D)$ holds, the endings of interior segments of C may also be in $B^+(D)$. In the algorithms below, we wish to treat endings of segments $I(C)$ similarly to segments of $I(D)$ up to a certain step, and later we wish to treat them as segments $B(D)$. We determine how to handle segments of $B(D) \setminus F$ by the set F^* defined below.

For any $F \subset B^+(C)$, we define F^* as follows. Let w_F be the vertex of C with the largest y-coordinate. Now let $\partial_F C \subset \partial C$ be the shortest polygonal path along the boundary ∂C of C containing all endpoints of F on ∂C, but not containing w_F. Let F^* be the set of segments in $B(C) \setminus F$ which have one endpoint in $\partial_F C$ (and the other endpoint in $\mathrm{int}(C)$). (See Figure 5.)

PROPOSITION 8. *Given $F \subset B^+(C)$, if $B(C) \setminus F^*$ contains no convex cycle, then there is a convex sequence $\mathcal{S} = (\ell_1, \ell_2, \dots, \ell_t)$, $t \in \mathbb{N}$, of $B(C) \setminus F^*$ such that $p(\ell_1) \in \partial C \setminus \partial_F C$ or $g(\ell_t) \in \partial C \setminus \partial_F C$.*

Proof. If there is a segment $\ell \in F$ such that $g(\ell)$ is in $\partial C \setminus \partial_F C$, then let $\mathcal{S} = (\ell)$. Assume that $g(\ell) \notin \partial C \setminus \partial_F C$ for any $\ell \in F$. Let $X = B(C) \setminus F^*$.

Let $p(r_1)$ and $p(s_1)$ be the first and last points of the curve $\partial_F C$ (with possibly $p(r_1) = p(s_1)$ if $|F| = 1$). Suppose w.l.o.g. that the y-coordinate of $p(s_1)$ is greater than or equal to that of $p(r_1)$. Consider the sequence $\mathcal{S} = (s_1, s_2, \dots, s_t)$ starting with s_1. Notice that if $s_i \in F$ for the indices $i = 1, 2, \dots, j$, then the y-coordinates of $p(s_1), g(s_1), g(s_2), \dots, g(s_j)$ are strictly increasing. If $g(s_j)$ is on the boundary of C (i.e., $t = j$), then it is in $\partial C \setminus \partial_F C$ by the choice of s_1. Therefore, the sequence starting with s_1 must contain a segment $s_q \in X \setminus F$, $j < q \le t$. Let s_q be the last segment of $\mathcal{S}$ from the set $X \setminus F$. Finally let $t' = t - q + 1$ and let $\mathcal{S}'$ be the post-fix sequence $(s_q, s_{q+1}, \dots, s_t)$ of $\mathcal{S}$.

We know that $p(s_q) \in \partial C \setminus \partial_F C$; now we show that $\mathcal{S}'$ is convex. Denote the elements of $\mathcal{S}'$ by $(\ell'_1, \ell'_2, \dots, \ell'_{t'})$. Every sequence of one or two segments is convex, so assume that $t' \ge 3$. For every segment $\ell \in B(C)$, we distinguish a left and a right side such that the closed curve on the boundary of C oriented in counterclockwise order

traverses ℓ from left to right. We may suppose w.l.o.g. that the extension of ℓ'_1 hits the left side of ℓ'_2. If the extension of every ℓ'_i, $i = 2, 3, \ldots, t'-1$, hits the left side of ℓ'_{i+1}, then S' is convex. Suppose, to the contrary, that $j \in \{2, 3, \ldots, t'-1\}$ is the first index such that ℓ'_j hits the right side of ℓ'_{j+1}. The polygonal curve $p(\ell'_j)g(\ell'_j)p(\ell'_{j+1})$ dissects C into two parts: C_1 and C_2. One part, say C_1, contains $\ell'_1 \in X \setminus F$ and all of the segments $\ell'_1, \ell'_2, \ldots, \ell'_{j-1}$, because X contains no convex cycle. Since both ℓ'_j and ℓ'_{j+1} are in F and $g(\ell'_{t'})$ lies in $\partial C_2 \cap \partial C$, the common boundary of C and C_1 belongs to $\partial_F C$. Therefore $p(\ell'_1) \in \partial_F C$, which in turn implies that ℓ'_1 is in F, a contradiction. □

PROPOSITION 9. *Given a cell C, a set of segments L, and a nonempty set $F \subset B^+(C)$, at least one of the following three statements holds.*

(a) *There is a convex sequence $\mathcal{S} = (\ell_1, \ell_2, \ldots, \ell_t)$, $t \in \mathbb{N}$, in $B(C) \setminus F^*$ such that $p(\ell_1) \in \partial C \setminus \partial_F C$ or $g(\ell_t) \in \partial C \setminus \partial_F C$.*

(b) *There is a convex cycle $\mathcal{S} = (\ell_1, \ell_2, \ldots, \ell_t)$, $t \in \mathbb{N}$, in $B(C) \setminus F^*$, and there is an index i, $0 \le i \le t$, such that F has no element in regions of $\pi(\mathcal{S}, i)$.*

(c) *There is a convex cycle $\mathcal{S} = (\ell_1, \ell_2, \ldots, \ell_t)$, $t \in \mathbb{N}$, in $B(C) \setminus F^*$ where $\ell_i, \ell_{i+1}, \ldots, \ell_j \in F$, $\ell_{j+1}, \ell_{j+2}, \ldots, \ell_{i-1} \notin F$, and the supporting line of ℓ_j intersects $p(\ell_{i-1})g(\ell_{i-1})$.*

Proof. If X contains no convex cycle, then (a) holds by Proposition 8. Suppose that X contains a convex cycle $\mathcal{S} = (\ell_1, \ell_2, \ldots, \ell_t)$.

If there is an index i such that every segment of F is either part of the cycle $\mathcal{S}$ or within one of two regions S_i and S_{i+1}, then (b) holds by Remark 1. Assume that segments of F are in at least three consecutive regions $S_{i-1}, S_i, S_{i+1}, \ldots, S_j$, where $j \neq i-1, i$. Consequently, $\ell_i, \ell_{i+1}, \ldots, \ell_j \in F$ and $\ell_{j+1}, \ell_{j+2}, \ldots, \ell_{i-1} \notin F$. Now, if the supporting line of ℓ_j does not intersect $p(\ell_{i-1})g(\ell_{i-1})$, then $\pi(\mathcal{S}, j) \subset \{S_{j+1}, S_{j+2}, \ldots, S_{i-2}\}$. Thus F has no element in regions of $\pi(\mathcal{S}, j)$ and (b) holds. Otherwise (c) holds. □

Proposition 9 states that at least one of (a), (b), and (c) holds, but it does not provide us with the convex sequence or convex cycle $\mathcal{S}$. It is easy to find an $\mathcal{S}$ by shooting a ray from an initial segment of X and iterating ray shooting queries until we hit the boundary of the cell or hit a previous (extended) segment. In the latter case, we identify a convex cycle, which necessarily satisfies (a) or (b). An algorithm finding any one convex sequence or cycle $\mathcal{S}$ may require a linear number of ray shooting queries among the segments of X, even if $\mathcal{S}$ consists of a small number of segments. Here, we do not elaborate on how to find an appropriate sequence or cycle $\mathcal{S}$, since in the main partition algorithms (Algorithms 5 and 6 below) we need to find an appropriate $\mathcal{S}$ possibly $\Omega(n)$ times, and we refer to a data structure appropriate for that many ray shooting queries at the end of section 5.

5. BSP for boundary segments. We demonstrate Lemma 1(i) in the following refined formulation (Lemma 10), which is necessary for the induction step in our proof.

DEFINITION 5. *We say that the directions of a subset of segments $F \subset B^+(C)$ are* consecutive *if the directions are consecutive in the linear order determined by the minimal angle α, $0 < \alpha < \pi$, made with the x-axis. (Recall that none of the segments of L are horizontal.)*

LEMMA 10. *Let L and C be as in Lemma 1. Consider a nonempty set $F \subset B^+(C)$ such that the directions in F belong to f consecutive directions. There is a BSP for F such that*

1. *every* $\ell \in B(C) \setminus F^*$ *is cut at most* $3\lfloor \log_2 f \rfloor + 3$ *times,*
2. *every* $\ell \in F^*$ *is cut at most* $3\lfloor \log_2 f \rfloor + 8$ *times,*
3. *every* $\ell \in I(C)$ *is cut at most* $6\lfloor \log_2 f \rfloor + 11$ *times.*

We prove Lemma 10 by giving an explicit partition algorithm (Algorithm 5) and showing that the number of cuts during that algorithm is within the bounds of Lemma 10. Throughout the remainder of this paper, we use the shorthand notation $F \cap D = \{\ell \cap \operatorname{int}(D) : \ell \in F\} \setminus \{\emptyset\}$ for the portions of the segments of $F \subset B^+(C)$ in a subcell $D \subset C$.

ALGORITHM 5. *Input:* $[C, L(C), F]$ *where* C *is a cell,* L *is a set of segments, and* $F \subset B^+(C)$, $F \neq \emptyset$.

(1) *Let* $X := B(C) \setminus F^*$.
(2) *Find a convex sequence or a convex cycle* $\mathcal{S}$ *as in Proposition* 9.
(3) *In the three cases of Proposition* 9 *proceed as follows:*
   (a) *Apply* $P(\mathcal{S})$ *for the convex sequence* $\mathcal{S}$.
   (b) *Apply* $P^0(\mathcal{S}, i)$ *for the convex cycle* $\mathcal{S}$.
   (c) *If the segments of* F *crossed by* $p(\ell_i)p(\ell_{i-1})$ *belong to at most* $\lfloor f/2 \rfloor$ *consecutive directions,*
      – *then dissect* C *along* $p(\ell_i)p(\ell_{i-1})$ *and apply* $P^0(\mathcal{S}, j)$;
      – *else dissect* C *along* $p(\ell_j)p(\ell_{i-1})$ *and apply* $P^0(\mathcal{S}, j-1)$.
(4) *Apply all possible free cuts.*
(5) *Call recursively Algorithm* 5 *for* $[D, L(D), F \cap D]$ *in every resulting cell* D *where* $F \cap D$ *is nonempty.*

In step (3)(c), the line segment $p(\ell_i)p(\ell_{i-1})$ dissects the region S_{i-1} and the line segment $p(\ell_j)p(\ell_{i-1})$ dissects the regions $S_j, S_{j+1}, \ldots, S_{i-2}$. The line through ℓ_j separates $p(\ell_{i-1})$ from $g(\ell_{i-1})$ and from the region S^*. This implies that the segment $p(\ell_j)p(\ell_{i-1})$ does not intersect S^* but crosses all the segments $\ell_{j+1}, \ell_{j+2}, \ldots, \ell_{i-1}$. Therefore $\mathcal{S}$ remains a convex cycle after a dissection along either $p(\ell_i)p(\ell_{i-1})$ or $p(\ell_j)p(\ell_{i-1})$, so $P^0(\mathcal{S}, j)$ and $P^0(\mathcal{S}, i)$ are defined.

Proof of Lemma 10. We proceed by induction on $|L(C)|$. Our lemma clearly holds for $|L(C)| = 1$. Suppose that it holds for every $|L'(C')| < |L(C)|$. Put $X = B(C) \setminus F^*$. We show that Algorithm 5 provides a required BSP.

Notice that segments are cut only in step (3) of Algorithm 5. The free cuts in step (4) do not increase the number of cuts on segments. We use one common argument for cases (a) and (b); while for case (c) we use another argument. In the first two cases, the partition $P(\mathcal{S})$ (and, resp., $P^0(\mathcal{S}, i)$) has, as we show below for both cases, the following two properties:

(α) If a segment $\ell \in X$ is cut, then it is cut at most twice and is either part of $\mathcal{S}$ or its ending is in a subcell D of $\mathcal{R}(P(\mathcal{S}))$ (or, resp., $\mathcal{R}(P^0(\mathcal{S}, i))$) in which $F \cap D$ is empty.
(β) If a segment $\ell \in I(C) \cup F^*$ is cut, then it is cut at most four times and its ending(s) are in sets $B(D) \setminus (F \cap D)^*$ in the corresponding subcell(s) D of $\mathcal{R}(P(\mathcal{S}))$ (or, resp., $\mathcal{R}(P^0(\mathcal{S}, i))$).

We prove properties (α) and (β) below, but let us first show that they establish our lemma: If a segment of X is cut, then it is cut at most twice and its ending is cut no more in the induction step by (α). If a segment of F^* is cut, then it is cut at most four times and its ending will be cut, by (β) and by induction, at most $3\lfloor \log_2 f \rfloor + 3$ more times, which totals to at most $3\lfloor \log_2 f \rfloor + 7$ cuts. If a segment of $I(C)$ is cut, then it is cut at most four times and either of its endings will be cut, by (β) and by induction, at most $3\lfloor \log_2 f \rfloor + 3$ more times, which totals to at most $6\lfloor \log_2 f \rfloor + 10$

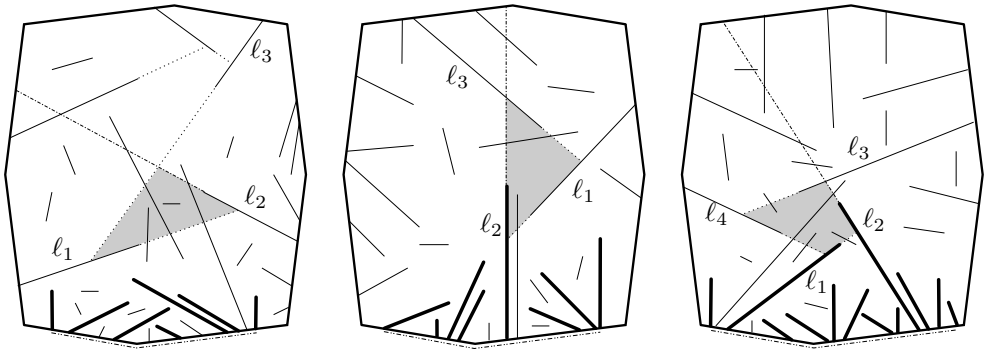


FIG. 6. *Case* (b) *in Algorithm* 5 *with* $i = 2$. *Segments of* F *are bold.*

cuts.

Case (a). By Proposition 2, no segment of $X = B(C) \setminus F^*$ is cut, so (α) follows. We need to prove (β). (α) implies that in any subcell $D \in \mathcal{R}(P(\mathcal{S}))$ the common point of an $\ell \in F \cap D$ and ∂D is necessarily on $\partial_F C$. Furthermore, a common point of an $\ell \in I(C) \cup F^*$ cut by $P(\mathcal{S})$ and ∂D is on $\mathcal{T}(P(\mathcal{S}))$. Therefore to prove (β), it is enough to show that $\partial_F C \cap \partial D$ is a connected curve for every subcell $D \in \mathcal{R}(P(\mathcal{S}))$. But $\partial_F C \cap \partial D$ is necessarily connected, since $\mathcal{T}(P(\mathcal{S}))$ is connected and contains at least one point (i.e., $p(\ell_1)$ or $g(\ell_t)$) from $\partial C \setminus \partial_F C$.

Case (b). Every segment of F either is part of the convex cycle $\mathcal{S}$ or lies in a cell S_j, $S_j \in \bar{\pi}(\mathcal{S}, i)$, of $\mathcal{R}(P^0(\mathcal{S}, i))$. Therefore, by Propositions 5 and 6, a segment of F can be cut (at most once) only if it is in the cycle $\mathcal{S}$. By Proposition 5, if any $\ell' \in B(C) \setminus (F \cup F^*)$ is cut, then its ending must lie in some region S_j, $j \in \pi(\mathcal{S}, I)$. This proves (α). Finally, as in case (a), (β) follows from the fact that $\mathcal{T}(P^0(\mathcal{S}, i))$ is connected and, by Proposition 4, contains at least one point from $\partial C \setminus \partial_F C$. (See Figure 6 for illustrations.)

Case (c). Observe that the sets of directions of segments of F which cross the segments $p(\ell_i)p(\ell_{i-1})$ and $p(\ell_j)p(\ell_{i-1})$, respectively, are disjoint: Segments of $B^+(C)$ cannot cross both $p(\ell_i)p(\ell_{i-1})$ and $p(\ell_j)p(\ell_{i-1})$. Indeed, the directions[2] of rays along segments of X crossing both $p(\ell_i)p(\ell_{i-1})$ and $p(\ell_j)p(\ell_{i-1})$ are between the directions of $\overrightarrow{p(\ell_{i-1})p(\ell_i)}$ and $\overrightarrow{p(\ell_{i-1})p(\ell_j)}$. The opposite of these directions lies between $\overrightarrow{p(\ell_i)p(\ell_{i-1})}$ and $\overrightarrow{p(\ell_j)p(\ell_{i-1})}$, which is a proper subset of the directions between $\overrightarrow{\ell_i}$ and $\overrightarrow{\ell_j}$. Recall that the directions between the directions of $\overrightarrow{\ell_i}$ and $\overrightarrow{\ell_j}$ are among the consecutive directions of $F \subset B^+(C)$. Therefore segments of X with opposite directions must belong to $B^-(C)$. (See Figure 7 for an illustration.)

In step (3)(c) of Algorithm 5, if the segments of F crossed by $p(\ell_i)p(\ell_{i-1})$ belong to at most $\lfloor f/2 \rfloor$ consecutive directions, then C is dissected along $p(\ell_i)p(\ell_{i-1})$; if the segments of F crossed by $p(\ell_j)p(\ell_{i-1})$ belong to at most $\lfloor f/2 \rfloor$ consecutive directions, then C is dissected along $p(\ell_j)p(\ell_{i-1})$. Suppose w.l.o.g. that the first case is called and our algorithm dissects C along $p(\ell_i)p(\ell_{i-1})$ and applies $P^0(\mathcal{S}, j)$. Let $\mathcal{R}(Q)$ denote the set of the resulting subcells. Instead of properties (α) and (β), we have the following two properties:

(γ) If a segment of X is cut, then it is cut at most three times and either is part

[2]The direction $\alpha \in [0°, 180°)$ of a line corresponds to two opposite directions, α and $\alpha + 180°$, of rays.

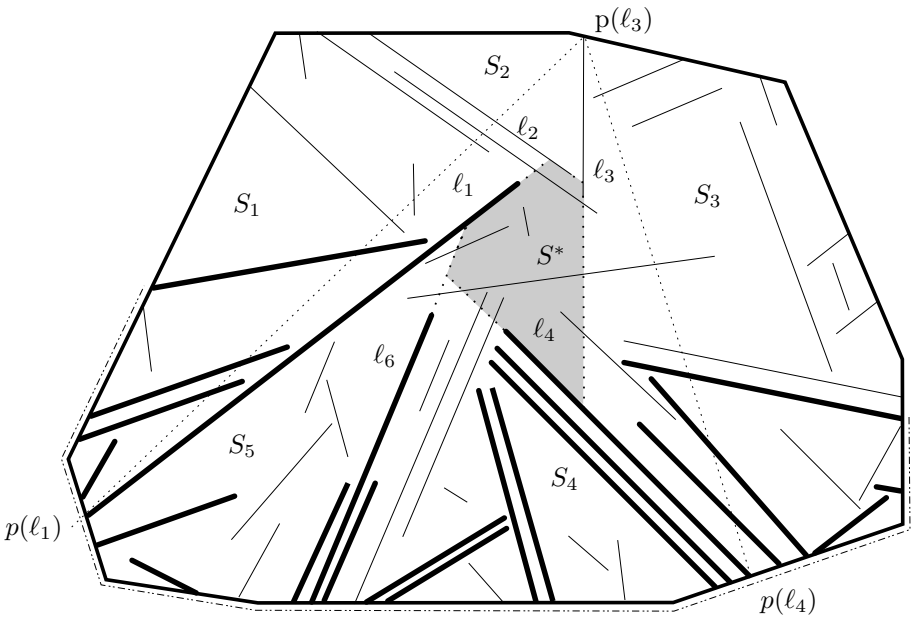


FIG. 7. *Example for case* (c), *with a convex cycle highlighted and with* $i = 4$ *and* $j = 1$.

of $\mathcal{S}$ or its ending is in a subcell $D \in \mathcal{R}(Q)$ in which $F \cap D$ belongs to at most $\lfloor f/2 \rfloor$ consecutive directions.

(δ) A segment of $F^* \cup I(C)$ can be cut in two possible ways:
– Either it is cut at most five times and each of its endings is in some $B(D) \setminus (F \cap D)^*$, $D \in \mathcal{R}(Q)$; or
– it is cut at most three times and one of its ending is in $(F \cap D)^*$, $D \in \mathcal{R}(Q)$, where segments of $F \cap D$ belong to at most $\lfloor f/2 \rfloor$ consecutive directions, and a second ending (for segments of $I(C)$) is in $B(D') \setminus (F \cap D')^*$, $D' \in \mathcal{R}(Q)$.

Let us first show that (γ) and (δ) establish our lemma: If a segment of X is cut, then it is cut at most three times. If $f > 1$, then its ending is further cut by induction at most $3\lfloor \log_2(f/2) \rfloor + 3$ more times, which totals to $3\lfloor \log_2 f \rfloor + 3$ cuts. If a segment of F^* is cut, then either it is cut at most five times and its ending is cut by induction at most $3\lfloor \log_2 f \rfloor + 3$ more times (this totals to $3\lfloor \log_2 f \rfloor + 8$ cuts) or it is cut at most three times and its ending is cut by induction at most $3\lfloor \log_2(f/2) \rfloor + 8$ more times (this totals to $3\lfloor \log_2 f \rfloor + 8$ cuts). If a segment of $I(C)$ is cut, then either it is cut at most five times and both its endings are cut at most $3\lfloor \log_2 f \rfloor + 3$ more times (this totals to $6\lfloor \log_2 f \rfloor + 11$ cuts) or it is cut at most three times and one ending is cut at most $3\lfloor \log_2 f \rfloor + 3$ more times and the other is cut at most $3\lfloor \log_2(f/2) \rfloor + 8$ times (this totals to $6\lfloor \log_2 f \rfloor + 11$ cuts, too).

Now we show (γ). Elements of X can be cut once by $p(\ell_{i-1})p(\ell_i)$ and at most twice by $P(\mathcal{S}, j)$. In particular, every segment of the convex cycle $\mathcal{S}$ can be cut at most twice only, and no segment ending remains. If a segment of $X \setminus \mathcal{S}$ is cut by $p(\ell_{i-1})p(\ell_i)$, then its ending is necessarily in a cell $D \in \mathcal{R}(Q)$ where all segments of $F \cap D$ are cut by $p(\ell_{i-1})p(\ell_i)$; therefore $F \cap D$ belongs to at most $\lfloor f/2 \rfloor$ consecutive directions. According to Proposition 6, $P^0(\mathcal{S}, j)$ can further dissect segments of X which lie in regions of $\pi(\mathcal{S}, j)$. Since only $S_{i-1} \in \pi(\mathcal{S}, j)$ contains segments of F, the BSP $P^0(\mathcal{S}, j)$ can dissect segments of X already cut by $p(\ell_{i-1})p(\ell_i)$. (Similarly if $P^0(\mathcal{S}, j-1)$ is

applied, we may assure that $g(\ell_{j-1}) \in p(\ell_{i-1})p(\ell_j)$; hence the line through ℓ_{j-1} cuts segments of F whose directions belong to $\lfloor f/2 \rfloor$ consecutive directions.)

For (δ), notice that an ending of a segment $\ell \in F^* \cup I(C)$ is in $(F \cap D)^*$ for some subcell $D \in \mathcal{R}(Q)$ only if its endpoint on ∂D is between two endpoints of two endings from $F \cap D$ which are cut by $p(\ell_{i-1})p(\ell_i)$ or by $P^0(\mathcal{S}, j)$. If this happens, then ℓ is cut at most once by $p(\ell_{i-1})p(\ell_i)$ and at most twice more only by $P^0(\mathcal{S}, j)$, and the second ending of an $\ell \in I(C)$ belongs to $B(E) \setminus ((F \cap E) \cup (F \cap E)^*)$ for an $E \in \mathcal{R}(Q)$. Otherwise $\ell \in F^* \cup I(C)$ can be cut five times: once by $p(\ell_{i-1})p(\ell_i)$ and four times by $P^0(\mathcal{S}, j)$, but the ending(s) do not belong to $(F \cap D)^*$ in the corresponding subcell(s) $D \in \mathcal{R}(Q)$. □

Our proof can easily be turned into an $O(n^2)$ time and $O(n^2)$ space algorithm. We expect that this complexity is not optimal. Our algorithm takes $O(n^2)$ time because finding a convex cycle $\mathcal{S} \subset X$ can use up to a linear number of ray shooting queries (even if $\mathcal{S}$ consists of a constant number of segments), and we cannot use the information from this computation in the induction steps, as many new elements are inserted into X for the subproblems. Each time a new segment ending ℓ is inserted into X, not only do we need to compute the segment its extension hits, but we also need to make updates on every segment of X whose extension now hits ℓ with respect to X. Computing the dissections alone could be done in $O(n^{4/3+2\varepsilon}k^{2/3+\varepsilon})$ time by ray shooting techniques of, e.g., Agarwal [1] and Agarwal and Matoušek [3]. It can offer a space-time trade-off, but the updates on the structure of X would require $O(n^3)$ queries.

Once we allow for an $O(n^2)$ time and space algorithm, the full arrangement of the n lines through the line segments is at hand. It can be computed in $O(n^2)$ time and space by the incremental algorithm of Chazelle, Guibas, and Lee [5] (see also [16, 15]). This data structure allows us to insert new lines (in particular, new elements of X and the dissecting lines which are not supporting lines of any segment) into the arrangement in linear time.

Using the notation introduced in the beginning of this section, we show the life cycle of a segment $\ell \in L$, as follows. A segment can, at first, be in $I(C)$. If ℓ is cut into pieces, then the two endings are in $B^+(C_1)$ and $B^-(C_2)$ (either in some set F^* or in $B(C) \setminus F^*$). Next, the two endings move independently into X and eventually to F. We can keep track of these events for each segment ending, and there is only a constant number of events for each segment.

We first compute the arrangement of the n supporting lines of the segments of L in $O(n^2)$ time. Then we compute the convex hull $\text{conv}(\bigcup L)$, the cell corresponding to the root of our BSP tree. We build a BSP by our Algorithm 1 while maintaining the sets $L(C)$, $I(C)$, $B^+(C)$, and $B^-(C)$ for each cell corresponding to a leaf of our current BSP tree. The total cost of maintaining $L(C)$ for all leaves is $O(n^2)$, since the cell containing a segment endpoint can be dissected at most linearly many times. In each round of Algorithm 1, we need to recompute X and the function $g(\ell)$ for every segment $\ell \in X$: For each segment ending we can find the hitting point of its extension in linear time using the full arrangement. A segment ending can occur in X in at most two more rounds of Algorithm 1. In the arrangement of the segments, we label each edge maintaining the information whether it is

- a portion of a segment in $L(C)$, and in particular belongs to $I(C)$, $B^+(C)$, or $B^-(C)$;
- part of a segment ending in F, F^*, or X;
- part of the boundary of a cell C; and

– part of an extension of a segment of X to another segment of X or to ∂C.

Thus we can maintain a pointer from each segment ending in X to the point hit by its extension at another element of X or at ∂C. When a segment of $\ell \in I(C)$ is cut and its ending is inserted into X in step (1) of Algorithm 5, we can update the information on the extensions of elements of X in linear time: at each intersection of ℓ with an extension of a previous segment $\ell' \in X$, we simply update the pointer of ℓ' to ℓ and delete the labels of the edges on the line through ℓ' beyond the intersection with ℓ. The total time complexity of maintaining hitting relations within X amounts to $O(n^2)$. Relying on these pointers, we find either a convex cycle or a convex sequence complying with one of the three cases of Proposition 9 in linear time (that is, we can find a sequence or cycle $\mathcal{S}_C$ in each cell $C \in \mathcal{R}_i$ in $O(|L(C)|)$ time, which totals to $O(n)$ time for all cells of the current partition). In either case, we dissect a cell C into subcells along the convex cycle or convex sequence and eliminate at least one segment ending from X. The total time spent on searching cycles is therefore $O(n^2)$, too.

The space complexity can be reduced to $O(n)$ by allowing $O(n^2 \log n)$ time. For each segment $\ell \in L$ we maintain the information if it is in the interior of any cell of the current partition or if ℓ is already dissected and its two endings belong to $B^+(C_1)$ and $B^-(C_2)$ and the middle of ℓ is the boundary of some cells of the current partition. Once a segment ℓ is dissected, we also maintain the two extensions of the endings to another segment of X or to the boundary of a cell. This involves a ray shooting query for the first time. We invest $O(n \log n)$ time into every ray shooting query: We spend $O(n)$ time on computing the intersection with all other supporting lines of segments and cutting lines and $O(n \log n)$ time on sorting the intersections. When a segment ending ℓ is inserted into X, we can update the information on the extensions of other segments of X (which might hit ℓ from now on) in $O(n \log n)$ time, too, based on the sorted list of intersections of ℓ with extensions of other segments in X. Similarly, we use $O(n \log n)$ time for a dissection along a segment. As each segment ending implies one ray shooting, one insertion into X, and one cutting, the total time amounts to $O(n^2 \log n)$.

6. Autopartition for boundary segments. In this section , we formulate and prove a stronger version of Lemma 1(ii). The proof follows the scheme of section 5.

LEMMA 11. *Let C and L be as in Lemma 1. Consider a nonempty set $F \subset B^+(C)$ such that the directions in F belong to f consecutive directions. There is a BSP P for F such that*

1. *every segment of* F *is cut at most* $2f$ *times,*
2. *every segment of* F^* *is cut at most* $4f+4$ *times,*
3. *every segment of* $I(C)$ *is cut at most* $4f+7$ *times,*
4. *every segment of* $B(C) \setminus (F \cup F^*)$ *is cut at most once.*
5. *In particular, if an ending of a segment $\ell \in F^*$ or $\ell \in I(C)$ is in a common subcell with an ending of segment of $B(C) \setminus (F \cup F^*)$, then ℓ is cut at most 5 or 7 times, respectively.*
6. *Every region corresponding to an internal node $v \in P$ is dissected along a segment of $L(R_v)$.*

We prove Lemma 11 by giving an explicit partition algorithm (Algorithm 6), which is a modified version of our Algorithm 5. Specifically, Algorithm 6 has new elements only in step (3)(c). The proof of Lemma 11 is based on the same case distinction as the proof of Lemma 10. The only difference is in the argument for case (c), where we are bound to use dissections along segments of $L(C)$. For the proof of Lemma 10, we applied induction to sets $F \cap D$ in the subcells of C. When segments

of $F \cap D$ were cut in step (3)(c), we ensured that their endings belonged to at most $\lfloor f/2 \rfloor$ consecutive directions. If we had to use autopartitions, we were unable to find a scheme which halved the number of consecutive directions. We can only ensure that if the segments of $F \cap D$ are cut in one loop, then the number of their consecutive directions decreases by a constant.

ALGORITHM 6. *Input:* $[C, L(C), F]$ *where* C *is a cell,* L *is a set of segments, and* $F \subset B^+(C)$, $F \neq \emptyset$.

(1) *Let* $X := B(C) \setminus F^*$.
(2) *Find a convex sequence or a convex cycle* $\mathcal{S}$ *as in Proposition* 9.
(3) *In the three cases of Proposition* 9 *proceed as follows:*
  (a) *Apply* $P(\mathcal{S})$ *for the convex sequence* $\mathcal{S}$.
  (b) *Apply* $P^0(\mathcal{S}, i)$ *for the convex cycle* $\mathcal{S}$.
  (c) (i) *If the line through* ℓ_{i-1} *does not cross the segment* ℓ_1 *but crosses the extended segment* $p(\ell_1)g(\ell_1)$, *then apply* $P^1(\mathcal{S}, i-1)$; *else apply* $P^0(\mathcal{S}, i-1)$.
    (ii) *Apply all possible free cuts.*
    (iii) *In every resulting cell* D, *let* $F_x(D)$ *denote the segments of* $F \cap D$ *whose endpoint lies on* $\partial_F C$.
    (iv) *Call recursively Algorithm* 6 *for* $[D, L(D), F_x(D)]$ *in every cell where* $F_x(D) \neq \emptyset$.
(4) *Apply all possible free cuts.*
(5) *Call recursively Algorithm* 6 *for* $[D, L(D), F(D)]$ *in every resulting cell* D *where* $F(D)$ *is nonempty.*

Proof of Lemma 11. We proceed by induction on $|L(C)|$. Our lemma clearly holds for $|L(C)| = 1$. Suppose that it holds for every $|L'(C')| < |L(C)|$. For cases (a) and (b), we can repeat the argument of Lemma 10 with the parameters in 1–4 of our Lemma 11. We only need to remark that properties 5 and 6 of Lemma 11 hold by induction.

Let us recall the setting in case (c): We assume that we are given a cell C, a set of segments L, and a subset $F \subset B^+(C)$ of the boundary segments, and that $X = B(C) \setminus F^*$. We assume, furthermore, that there is a convex cycle $\mathcal{S} \subset X$ and that there are at least two common segments of $\mathcal{S}$ and F. We denote the common segments by $\ell_i, \ell_{i+1}, \ldots, \ell_j$ appearing in this (cyclic) order in $\mathcal{S}$. The supporting line of ℓ_j intersects $p(\ell_{i-1})g(\ell_{i-1})$, which implies that the direction of the line ℓ_{i-1} is between the consecutive directions of F between ℓ_i and ℓ_j. (See Figure 8.)

In step (3)(c)(i) of Algorithm 6 a BSP $P(\mathcal{S}, i-1)$ is applied. Let s denote the line through ℓ_{i-1}. Line s may cut the regions in $\pi(\mathcal{S}, i-1)$, and $\pi(\mathcal{S}, i-1) \subseteq \{S_i, S_{i+1}, \ldots, S_{j-1}\}$. By Proposition 1, it cannot cut any segment of $B(C) \setminus (F \cup F^*)$.

In every subcell $D \in \mathcal{R}(P(\mathcal{S}, i-1))$, we partition the endings of the segments in $F \cap D$ into two sets: Let $F_s(D)$ be the subset of endings of segments cut by s, and let $F_x(D)$ be the subset of segments not cut by s. Observe that segments in $F_s(D)$ have at most $f-1$ consecutive directions: The direction of s cannot occur in $F_s(D)$. Observe, furthermore, that $F_s(D)^*$ and $F_x(D)^*$ are disjoint in every subcell $D \in \mathcal{R}(P(\mathcal{S}, i-1))$. Elements of $F_x(D)^*$ all belong to F^*. Elements of $F_s(D)^*$ are endings of segments from F^* or $I(C)$ cut by the line s.

In step (3)(c)(iv) of Algorithm 6, we obtain a BSP P_D for $F_x(D)$ in each subcell $D \in \mathcal{R}(P(\mathcal{S}, i-1))$. These BSPs jointly partition C into a collection $\mathcal{E}$ of subcells. In step (5) of Algorithm 6, a BSP for $(F \cap D) \cap E$ (or, for short, $F \cap E$) is applied in each subcell $E \in \mathcal{E}$, $D \in \mathcal{R}(P(\mathcal{S}, i-1))$. Notice that $F \cap E = F_s(D) \cap E$ since a

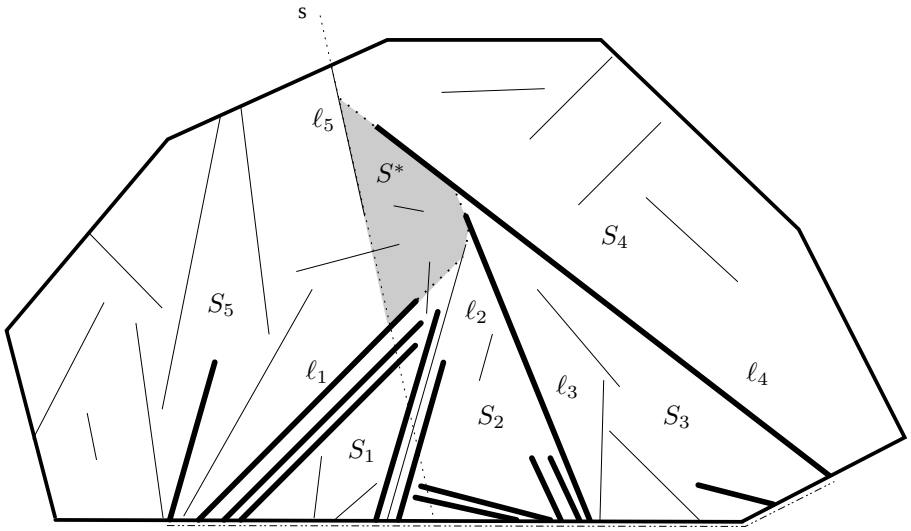


FIG. 8. *Example for case* (c), *with a convex cycle highlighted and with* $i = 1$ *and* $j = 4$.

BSP for $(F_x \cap D)$ was already applied. For brevity, we refer to the three partitioning schemes as $P(\mathcal{S}, i-1)$, P_x, and P_s. (The last two recursive calls can be carried out in parallel in the nonoverlapping subcells.)

We check one by one statements 1–5 of our lemma. Statement 6 follows automatically.

1. If a segment $\ell^1 \in F$ is not cut by s, then it belongs to $F_x(D)$, $D \in \mathcal{R}(P(\mathcal{S}, i-1))$ and is cut at most $2f$ times by induction. If $\ell^1 \in F$ is cut by s, then P_x (called in step (3)(c)(iv) for $F_x(D)$ in a subcell $D \in \mathcal{R}(P(\mathcal{S}, i-1))$) cuts it at most once more, and P_s (by induction to $F_s(E)$, $E \subset D$) cuts it at most $2(f-1)$ times.

2. A segment $\ell^2 \in F^*$ is cut at most 4 times by $P(\mathcal{S}, i-1)$ by Propositions 6 and 7. If the ending of ℓ^2 is in $F_s(D)^*$, $D \in \mathcal{R}(P(\mathcal{S}, i-1))$, then it is cut at most once by P_x and at most $4(f-1)+4$ times by P_s. This totals to $4f+4$ cuts. If the ending of ℓ^2 is in $F_x(D)^*$, $D \in \mathcal{R}(P(\mathcal{S}, i-1))$, then it was not cut by $P(\mathcal{S}, i-1)$ and it is cut at most $4f+4$ times by P_x. It can be cut once more by P_s, but then by Lemma 11(5) it was cut at most 5 times by P_x; this totals to at most 6 cuts. Otherwise if $\ell^2 \in F^*$ is cut by $P(\mathcal{S}, i-1)$ and its ending is in $B(D) \setminus (F_x(D) \cup F_x(D)^*)$ for some $D \in \mathcal{R}(P(\mathcal{S}, i-1))$ and in $B(E) \setminus ((F_s \cap E) \cup (F_s \cap E)^*)$, $E \subseteq D$, then it is cut at most 4 times by $P(\mathcal{S}, i-1)$, at most once by P_x, and at most once by P_s, that is, at most 6 times in total.

3. A segment $\ell^3 \in I(C)$ is cut at most 4 times by $P(\mathcal{S}, i-1)$. Note that if ℓ^3 is cut by s, then at most one of its endings may be in $F_s(D)^*$, $D \in \mathcal{R}(P(\mathcal{S}, i-1))$. If one ending is actually in $F_s(D)^*$, then this ending is cut at most once by P_x and at most $4(f-1)+4$ times by P_s, while the other ending is cut at most once by both P_x and P_s. This implies at most $4f+7$ cuts on ℓ^3. If neither ending is in $F_s(D)^*$, $D \in \mathcal{R}(P(\mathcal{S}, i-1))$, then P_x and P_s cuts either ending at most once each, which totals to at most 8 cuts on ℓ^3. If ℓ^3 is not cut by $P(\mathcal{S}, i-1)$ and, therefore, belongs to $I(D)$, $D \in \mathcal{R}(P(\mathcal{S}, i-1))$, then P_x cuts it at most $4f+7$ times. The subsequent BSP P_s can cut either of its endings at most once more, but in that case, by Lemma 11(5), it was cut at most 7 times by P_x; thus we obtain a total of at most 9 cuts.

4. A segment $\ell^4 \in B(C) \setminus (F \cup F^*)$ is cut by s only if ℓ^4 is part of $\mathcal{S}$. Moreover it cannot be in any of the regions $\pi(\mathcal{S}, i-1) \subseteq \{S_i, S_{i+1}, \ldots, S_{j-1}\}$, which are possibly cut by s. Therefore P_x cuts ℓ^4 at most once, but P_s cannot cut the ending of ℓ^4.

5. Observe that the BSP P_s for $F_s \cap E$ was applied only in subcells $E \in \mathbf{E}$ where $F_s \cap E$ is nonempty, that is, only in subcells of $\mathcal{R}(P(\mathcal{S}, i-1))$ which lie along the extension of segment ℓ_{i-1} beyond $g(\ell_{i-1})$. If a segment $\ell^5 \in F^* \cup I(C)$ is cut by $P(\mathcal{S}, i-1)$ (at most 4 times) and one of its endings lies in a cell $D_1 \in \mathcal{R}(P(\mathcal{S}, i-1))$ which has no common boundary with (the relative interior of) $\partial_F C$, then this ending is necessarily in $B(D_1) \setminus (F_x(D_1) \cup F_x(D_1)^*)$ and is cut at most once by P_x. P_s does not cut this ending, since $F_s \cap E = \emptyset$ for all such $E \subseteq D_1$. If, furthermore, $\ell^5 \in I(C)$ and its other ending lies in some $D_2 \in \mathcal{R}(P(\mathcal{S}, i-1))$, then it is in $B(D_2) \setminus (F_x(D_2) \cup F_x(D_2)^*)$ and then in $B(E) \setminus ((F_s \cap E) \cup (F_s \cap E)^*)$, $E \subseteq D_2$; hence is cut at most once by P_x and at most once by P_s. The total number of cuts is therefore 5 and 7 if $\ell \in F^*$ and $\ell \in I(C)$, respectively. □

It is straightforward to turn this proof into an $O(n^2 k)$ time and $O(n^2)$ space algorithm using the arrangement of the supporting lines of all segments, or into an $O(n^2 k \log n)$ time and $O(n)$ space algorithm by maintaining, as in section 5, only constant information per segment. The crucial difference, compared to the algorithms at the end of section 5, is that a segment ending can be inserted into X and deleted from it $O(k)$ times: in case 5 of the previous proof, a segment ending can be in $B(D) \setminus (F_x(D) \cup F_x(D)^*) \subset X$ then in $(F_s(D) \cap E)^* \not\subset X$. Each time, when we apply induction with a smaller value of the parameter f, some of the segment endings can be deleted from X, and we need to recompute the relations between elements of X.

Acknowledgments. I am grateful to Günter Rote, Emo Welzl, and the anonymous referee for their many useful remarks and suggestions that allowed me to improve this paper.

REFERENCES

[1] P. K. Agarwal, *Ray shooting and other applications of spanning trees with low stabbing number*, SIAM J. Comput., 21 (1992), pp. 540–570.

[2] P. K. Agarwal, E. F. Grove, T. M. Murali, and J. S. Vitter, *Binary space partitions for fat rectangles*, SIAM J. Comput., 29 (2000), pp. 1422–1448.

[3] P. K. Agarwal and J. Matoušek, *Ray shooting and parametric search*, SIAM J. Comput., 22 (1993), pp. 794–806.

[4] S. Ar, B. Chazelle, and A. Tal, *Self-customized BSP trees for collision detection*, Comput. Geom. Theory Appl., 15 (2000), pp. 91–102.

[5] B. Chazelle, L. J. Guibas, and D. T. Lee, *The power of geometric duality*, BIT, 25 (1985), pp. 76–90.

[6] N. Chin and S. Feiner, *Near real-time shadow generation using BSP trees*, in Proceedings of the 16th Conference on Computer Graphics and Interactive Techniques (Boston, MA), SIGGRAPH, ACM Press, New York, 1989, pp. 99–106.

[7] Y. Chrysanthou and M. Slater, *Computing dynamic changes to BSP tree*, Comput. Graphics Forum, 11 (1992), pp. 321–332.

[8] N. Dadoun, D. G. Kirkpatrick, and J. P. Walsh, *The geometry of beam tracing*, in Proceedings of the 1st ACM Symposium on Computational Geometry (Baltimore, MD), ACM, New York, 1985, pp. 55–61.

[9] F. d'Amore and P. G. Franciosa, *On the optimal binary plane partition for sets of isothetic rectangles*, Inform. Process. Lett., 44 (1992), pp. 255–259.

[10] M. de Berg, *Linear size binary space partitions for uncluttered scenes*, Algorithmica, 28 (2000), pp. 353–366.

[11] M. de Berg, M. de Groot, and M. Overmars, *New results on binary space partitions in the plane*, Comput. Geom. Theory Appl., 8 (1997), pp. 317–333.

[12] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry. Algorithms and Applications*, 2nd ed., Springer-Verlag, Berlin, 2000.

[13] S. E. Dorwald, *A survey of object-space hidden surface removal*, Internat. J. Comput. Geom. Appl., 4 (1994), pp. 325–362.

[14] A. Dumitrescu, J. S. B. Mitchell, and M. Sharir, *Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles*, in Proceedings of the 17th ACM Symposium on Computational Geometry (Medford, MA), ACM, New York, 2001, pp. 141–150.

[15] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Monogr. Theoret. Comput. Sci. EATCS Ser. 10, Springer-Verlag, Berlin, 1987.

[16] H. Edelsbrunner, J. O'Rourke, and R. Seidel, *Constructing arrangements of lines and hyperplanes with applications*, SIAM J. Comput., 15 (1986), pp. 341–363.

[17] H. Fuchs, Z. M. Kedem, and B. Naylor, *On visible surface generation by a priory tree structures*, in Proceedings of the 7th Conference on Computer Graphics and Interactive Techniques (Seattle, WA), SIGGRAPH, ACM Press, New York, 1980, pp. 124–133.

[18] D. Gordon and S. Chen, *Front-to-back display of BSP trees*, IEEE Comp. Graphics and Appl., 11 (1991), pp. 79–85.

[19] B. Naylor, *Partitioning tree image representation and generation from* 3D *geometric models*, in Proceedings of the 1992 Graphics Interface Conference (Vancouver, BC), Canadian Human-Computer Communications Society, 1992, pp. 201–212.

[20] B. Naylor, J. Amanatides, and W. Thibault, *Merging BSP trees yields polyhedral set operations*, in Proceedings of the 17th Conference on Computer Graphics and Interactive Techniques (Dallas, TX), SIGGRAPH, ACM Press, New York, 1990, pp. 115–124.

[21] M. S. Paterson and F. F. Yao, *Binary space partitions for orthogonal objects*, in Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA), SIAM, Philadelphia, 1990, pp. 100–106.

[22] M. S. Paterson and F. F. Yao, *Efficient binary space partitions for hidden-surface removal and solid modeling*, Discrete Comput. Geom., 5 (1990), pp. 485–503.

[23] M. S. Paterson and F. F. Yao, *Optimal binary space partitions for orthogonal objects*, J. Algorithms, 13 (1992), pp. 99–113.

[24] R. A. Schumacker, R. Brand, M. Gilliland, and W. Sharp, *Study for Applying Computer-Generated Images to Visual Simulation*, Tech. Report, U.S. Air Force Human Resources Laboratory, Brooks Air Force Base, San Antonio, TX, 1969.

[25] K. Sung and P. Shirley, *Ray tracing with the BSP tree*, in Graphics Gems III, D. Kirk, ed., Academic Press, San Diego, CA, 1992, pp. 271–274.

[26] I. E. Sutherland, R. F. Sproull, and R. A. Schumacker, *A characterization of ten hidden-surface algorithms*, ACM Comput. Survey, 9 (1984), pp. 1–55.

[27] Cs. D. Tóth, *A note on binary plane partitions*, in Proceedings of the 17th ACM Symposium on Computational Geometry (Medford, MA), ACM, New York, 2001, pp. 151–156.

[28] G. Vanecek, *Brep-index: A multidimensional space partitioning tree*, Internat. J. Comput. Geom. Appl., 1 (1991), pp. 243–261.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 326–344

ARC-DISJOINT PATHS IN EXPANDER DIGRAPHS*

TOM BOHMAN† AND ALAN FRIEZE†

Abstract. Given a digraph $D = (V, A)$ and a set of κ pairs of vertices in V, we are interested in finding, for each pair (x_i, y_i), a directed path connecting x_i to y_i such that the set of κ paths so found is arc-disjoint. For arbitrary graphs the problem is $\mathcal{NP}$-complete, even for $\kappa = 2$.

We present a polynomial time randomized algorithm for finding arc-disjoint paths in an r-regular expander digraph D. We show that if D has sufficiently strong expansion properties and the degree r is sufficiently large, then *all* sets of $\kappa = \Omega(n/\log n)$ pairs of vertices can be joined. This is within a constant factor of best possible.

Key words. edge disjoint paths, expander digraphs, randomized algorithm

AMS subject classifications. 68Q25, 68W20

PII. S0097539701398582

1. Introduction. Given a (graph) digraph $D = (V, A)$ with n vertices and a set of κ pairs of vertices in V, we are interested in finding, for each pair (x_i, y_i), an (undirected) directed path connecting x_i to y_i, such that the set of κ paths so found is (edge) arc-disjoint. This is a classical problem in graph theory. See Frank [7] for a survey and Chapter 9.2 of the recent book on digraphs by Bang-Jensen and Gutin [2].

For undirected graphs, the related decision problem is in $\mathcal{P}$ for fixed κ (see Robertson and Seymour [22]) but is $\mathcal{NP}$-complete if κ is part of the input. For digraphs the situation is seemingly much worse. Fortune, Hopcroft, and Wyllie [6] showed that the related decision problem is $\mathcal{NP}$-complete, even when $\kappa = 2$.

For undirected graphs there have been positive results in the case of expanders. Peleg and Upfal [21] presented a polynomial time algorithm for the case where D is a (sufficiently strong) bounded degree expander graph and $\kappa \leq n^{\epsilon}$ for a small constant ϵ that depends on the expansion property of the graph. This result has been improved and extended by Broder, Frieze, and Upfal [4, 5], Frieze [8, 9], Leighton and Rao [17], and Leighton, Rao, and Srinivasan [18, 19]. In particular Frieze [9] showed that if D has sufficiently strong *edge* expansion properties and r is sufficiently large, then *all* sets of $\kappa = \Omega(n/\log n)$ pairs of vertices can be joined. This is within a constant factor of a simple upper bound. The purpose of this paper is to extend this result to digraphs.

In this paper we discuss r-regular digraphs. A digraph D is r-regular if every vertex has in-degree and out-degree r. Let d_μ be the median distance between pairs of vertices in the digraph D which has m arcs. Clearly, there exists a collection of $O(m/d_\mu)$ pairs of vertices that cannot be connected by arc-disjoint paths because such a collection of paths would require more arcs than all the arcs available. In the case of an r-regular expander, this absolute upper bound on κ is $O(n/\log n)$ (assuming r is independent of n). We show that if D has sufficiently strong *arc* expansion

*Received by the editors November 26, 2001; accepted for publication (in revised form) August 20, 2002; published electronically January 17, 2003.

http://www.siam.org/journals/sicomp/32-2/39858.html

†Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213 (tbohman@moser.math.cmu.edu, alan@random.math.cmu.edu). The work of the first author was supported in part by a grant from the NSA. The work of the second author was supported in part by NSF grant CCR-9818411.

properties and r is sufficiently large, then *all* sets of $\kappa = \Omega(n/\log n)$ pairs of vertices can be joined. This is therefore within a constant factor of the optimum. The precise definition of "sufficiently strong" is given after the theorem.

THEOREM 1. *Let $D = (V, A)$ be an n-vertex, r-regular digraph. Suppose that D is a sufficiently strong arc expander. Then there exist $\epsilon_1, \epsilon_2 > 0$ such that D has the following property: For all sets of pairs of vertices $\{(x_i, y_i) \mid i = 1, \ldots, \kappa\}$ satisfying*

(i) $\kappa = \lceil \epsilon_1 rn/\log n \rceil$, *and*

(ii) *for each vertex v, $|\{i : x_i = v\}|, |\{i : y_i = v\}| \le \epsilon_2 r$,*

there exist arc-disjoint paths in D, each of length $O(\log n)$, joining x_i to y_i for each $i = 1, 2, \ldots, \kappa$. Furthermore, there is a polynomial time randomized algorithm for constructing these paths. The constants ϵ_1, ϵ_2 depend only on certain expansion parameters α, β, γ defined below. They do not depend on n or r. (For example, conditions (2) *with $\epsilon = \alpha$,* (3), (9), *and* (12) *suffice.)*

Remark 1. The algorithm is similar to the algorithm of [9]. The difficulty in moving from graphs to digraphs has been with that part of the algorithm for graphs which was based on the rapid mixing of a random walk on expanders. Random walks on digraphs are not necessarily time reversible, and the steady state can be hard to determine. We therefore abandoned this approach and replaced it with a different random choice of path (in part, this random choice for digraphs uses the multicommodity flow results of Leighton and Rao [16]).

It will be observed that this new algorithm can substitute for that given in [9].

Remark 2. If D has sufficiently strong *vertex* expansion properties, then we can take $\kappa = \lceil \epsilon_1 rn/\log_r n \rceil$; see Remark 4 below.

Remark 3. It is perhaps worth remarking that regularity is not crucial to the result. One can easily extend the results to digraphs where in- and out-degrees are constrained to be in the interval $[r, ar]$, where $a \ge 1$ is an absolute constant. All we need is for r to be sufficiently large. The crucial property is strong expansion.

1.1. Preliminaries. In this section we state the definitions for the expanders we work with here, make some preliminary observations about such expanders, and precisely define the notion of "sufficiently strong" expansion needed for Theorem 1. We begin with some notational conventions.

Let $D = (V, A)$ be a digraph and let $n = |V|$. For $S, T \subseteq V$ let $A_D^+(S, T)$ be the set of arcs with tail in S and head in T; that is,

$$A^+(S,T) = A_D^+(S,T) = \{(u,v) \in A \mid u \in S, v \in T\} \quad \text{and} \quad d^+(S,T) = |A^+(S,T)|.$$

We define $A^-(S, T)$ similarly (i.e., $A^-(S, T) = A^+(T, S)$). We set

$$A^+(S) = \{(u,v) \in A \mid u \in S, v \notin S\} \quad \text{and} \quad d^+(S) = |A^+(S)|.$$

So, for example, we have $A^+(S) = A^+(S, V \setminus S)$. We define $A^-(S)$ and $d^-(S)$ analogously. Throughout the paper, when $\star$ is used as a subscript or superscript, it stands for $+$ or $-$. We abbreviate $A^\star(\{v\}, T)$ to $A^\star(v, T)$. Thus, for $v \in V$, $d_D^+(v)$ and $d_D^-(v)$ denote the out-degree and in-degree of v in D.

We now have the notation necessary to introduce expansion. We define expanders in terms of arc expansion (a weaker property than vertex expansion). For $S \subseteq V$ let $\Phi_S^\star = d^\star(S)/|S|$. The (arc) expansion $\Phi = \Phi(D)$ of D is defined by

$$\Phi = \min_{\substack{S \subseteq V \\ |S| \le n/2}} \min\{\Phi_S^+, \Phi_S^-\}.$$

A digraph $D = (V, A)$ is a θ-expander if for every set $S \subseteq V$, $|S| \leq n/2$, we have $d^\star(S) \geq \theta|S|$; in other words, D is a θ-expander if $\Phi(D) \geq \theta$. An r-regular digraph $D = (V, A)$ is called an (α, β, γ)-expander if for every set $S \subseteq V$,

$$d^\star(S) \geq \begin{cases} (1-\alpha)r|S| & \text{if } |S| \leq \gamma n, \\ \beta r|S| & \text{if } \gamma n < |S| \leq n/2. \end{cases}$$

We naturally assume that $\beta < 1 - \alpha$. By "sufficiently strong" in Theorem 1 we mean that β, γ are arbitrary and α is sufficiently small; in particular, we assume that conditions (2) with $\epsilon = \alpha$, (3), (9), and (12) hold. We also assume throughout that r and n are sufficiently large (but r is *not* a function of n). We have made no real attempt to optimize constants. Such digraphs exist; in particular, random regular digraphs are usually (α, β, γ)-expanders. (See [4] for the corresponding notion in undirected graphs.)

We conclude this section with some preliminary observations about expanders. Since $|A^\star(S, S)| + d^\star(S) = r|S|$ we see that, putting $in(S) = |A^+(S, S)| = |A^-(S, S)|$, in an (α, β, γ)-expander

$$in(S) \leq \alpha r|S| \qquad \text{when } |S| \leq \gamma n. \tag{1}$$

For a digraph $\Delta = (V', A')$ and a set $S \subseteq V'$ we define its out-neighbor set $N^+_\Delta(S)$ as

$$N^+_\Delta(S) = \{w \notin S : \exists v \in S \text{ such that } (v, w) \in A'\}.$$

Similarly, the in-neighbor set of S, $N^-_\Delta(S)$, is given by

$$N^-_\Delta(S) = \{w \notin S : \exists v \in S \text{ such that } (w, v) \in A'\}.$$

LEMMA 1. *Suppose that $D = (V, A)$ is an (α, β, γ)-expander and that $D' = (V', A')$ is a subdigraph of D of expansion at least θr where $\theta > \alpha$. Suppose $S \subseteq V'$. If $|S| \leq \frac{\gamma\alpha}{\theta} n$, then*

$$|N^\star_{D'}(S)| \geq \frac{\theta - \alpha}{\alpha}|S|.$$

Proof. Suppose that $|S| \leq \frac{\gamma\alpha}{\theta} n$ and $T = N^\star_{D'}(S)$ satisfies $|T| < \frac{\theta-\alpha}{\alpha}|S|$. Then

$$|S \cup T| < \left(1 + \frac{\theta - \alpha}{\alpha}\right)|S| = \frac{\theta}{\alpha}|S| \leq \gamma n.$$

But $S \cup T$ contains at least

$$\theta r|S| > \theta r \left(1 + \frac{\theta - \alpha}{\alpha}\right)^{-1} |S \cup T| = \alpha r|S \cup T|$$

arcs, which contradicts (1). ∎

2. The algorithm. The input to our algorithm is a sufficiently strong (α, β, γ)-expander digraph D and a set of pairs of vertices $\{(x_i, y_i) \mid i = 1, \ldots, \kappa\}$ satisfying the premises of Theorem 1. The output is a set of κ arc-disjoint paths, $P_1, \ldots, P_\kappa$, such that P_i connects x_i to y_i.

The algorithm has three phases. We begin in Phase 0 by splitting our graph into 13 arc-disjoint expanders $D_i = (V, A_i)$, $1 \leq i \leq 13$. These graphs will be used for

various purposes in Phases 1 and 2, and each path we construct will be a union of paths in the D_i's. Phase 1 consists mainly of applications of GenPaths, an algorithm that uses the expander property to naively connect pairs of vertices with paths of length $O(\log n)$ one at a time, deleting the arcs from a path as soon as it is used. Of course, this deletion of arcs may quickly destroy the expander property. To compensate for this problem, GenPaths "shrinks" the expanders in which it finds paths. The real work of GenPaths is in keeping as many vertices as possible "connected" to these shrinking expanders. Those pairs of vertices that are not connected with paths in Phase 1 (i.e., those pairs that contain a vertex whose connection with one of the "shrinking expanders" is lost) are handled in Phase 2. There are $O(\frac{n}{\log^4 n})$ such pairs. Loosely speaking, Phase 2 uses the multicommodity flow algorithm of Leighton and Rao to give a distribution on paths connecting the remaining pairs such that **whp** paths chosen at random with respect to this distribution are arc-disjoint.

2.1. Phase 0. We need an algorithm for splitting an (α, β, γ)-expander digraph into 13 expander digraphs. Algorithms for splitting undirected graphs are given in [4] and [10]. They are easily adapted to digraphs, and we outline an adaptation of the algorithm of [10] in Appendix A. In the appendix we prove the following.

Theorem 2. *Suppose we have*

$$\frac{r}{\log r} \geq 91\epsilon^{-2} \quad \text{and} \quad \beta \geq 65\epsilon^{-2}r^{-1}\log 2er \tag{2}$$

and that D is an r-regular (α, β, γ)-expander, r constant. Then there is a randomized polynomial time algorithm which, with probability at least $1 - \delta$, constructs $A_1, A_2, \ldots, A_{13}$ such that the arc expansion Φ_i of $D_i = (V, A_i)$ satisfies

$$\Phi_i \geq (1-\epsilon)\frac{\Phi}{13} - (\alpha + 2\epsilon)\, r$$

for $i = 1, 2, \ldots, 13$.

This theorem is useful only if Φ is at least a constant multiple of r and α is sufficiently small. This is the case discussed in this paper. The algorithm runs in $O(n^2 \ln n \log \delta^{-1})$ expected time. There is not enough time to verify that the algorithm succeeds. Instead, we simply assume it has and repeat the split if we fail to find the required paths.

We apply the algorithm of Theorem 2 with $\alpha = \epsilon$ and assume that

$$\beta > 156\alpha. \tag{3}$$

Setting

$$\beta_0 = \frac{\beta}{13} - 4\alpha > 8\alpha > 0, \tag{4}$$

each D_i satisfies

$$\Phi_i = \Phi(D_i) \geq \beta_0 r, \qquad \text{and} \tag{5}$$

$$\beta_0 r \leq d_i^\star(v) < r \qquad \text{for all } v \in V. \tag{6}$$

2.2. Phase 1. Phase 1 uses expanders D_1 through D_6. The centerpiece of Phase 1 is the algorithm GenPaths, which connects a large collection of *random*

pairs of vertices in an expander with arc-disjoint paths. Since the pairs x_i, y_i are arbitrary, GenPaths cannot be applied to them directly; we must reduce the problem of connecting the x_i's to the y_i's to the problem of connecting random pairs.

In order to produce random pairs we introduce three random sets of κ vertices: $\tilde{X}$, $\tilde{Y}$, and Z. In the initialization step of Phase 1, a network-flow technique is used to find a collection of arc-disjoint paths $\mathcal{P}^1 = \{P_i^1 : i = 1, \dots, \kappa\}$ from $X = \{x_1, \dots, x_\kappa\}$ to $\tilde{X}$ in expander D_1 such that the path P_i^1 starts at x_i for $i = 1, \dots, \kappa$. It is important to note that we have no control over which element of $\tilde{X}$ is at the end of the path P_i^1 (but there will be one path ending at x for each $x \in \tilde{X}$). This network-flow technique is also used in the initialization step of Phase 1 to find a collection of arc-disjoint paths $\mathcal{P}^6 = \{P_i^6 : i = 1, \dots, \kappa\}$ from $\tilde{Y}$ to $Y = \{y_1, \dots, y_\kappa\}$ in D_6 such that the endpoint of the path P_i^6 is y_i. After the initialization step, we take a random ordering of $\tilde{X}$: $\tilde{X} = \{\tilde{x}_1, \dots, \tilde{x}_\kappa\}$. Furthermore, we order $\tilde{Y}$ so as to respect the pairing of $\tilde{X}$ and $\tilde{Y}$ that is inherited from the collections $\mathcal{P}^1$ and $\mathcal{P}^6$; that is, we set $\tilde{Y} = \{\tilde{y}_1, \dots, \tilde{y}_\kappa\}$ so that if the endpoint of P_i^1 is $\tilde{x}_j$, then $\tilde{y}_j$ is the starting point of P_i^6. Thus, it remains to find a collection of arc-disjoint paths connecting the pairs $\{(\tilde{x}_i, \tilde{y}_i) : i = 1, \dots, \kappa\}$. Unfortunately this sequence of pairs of vertices is *not* truly random. This is a consequence of the fact that the pairing between $\tilde{X}$ and $\tilde{Y}$ is determined by a deterministic process (i.e., knowledge of some of the *pairs* from this collection may bias the distribution on the unknown pairs). We overcome this problem by introducing the third random set $Z = \{z_1, \dots, z_\kappa\}$. The sequences $\{(\tilde{x}_i, z_i) : i = 1, \dots, \kappa\}$ and $\{(z_i, \tilde{y}_i) : i = 1, \dots, \kappa\}$ are perfectly random sequences of pairs of vertices (as long as we view them separately). It remains to connect these two sequences of pairs of vertices with arc-disjoint paths. This is the work of the algorithm GenPaths.

The input to GenPaths is a pair of expanders, D_a and D_b, and a collection of pairs of vertices $\{(v_i, u_i) : i = 1, \dots, \kappa\}$ that is generated uniformly at random. The output of GenPaths is a collection of arc-disjoint paths from v_i to u_i for $i = 1, \dots, \kappa$ that use only the arcs from D_a and D_b. We apply GenPaths twice. For the first application we set $D_a = D_2$, $D_b = D_3$, and $\{(v_i, u_i) : i = 1, \dots, \kappa\} = \{(\tilde{x}_i, z_i) : i = 1, \dots, \kappa\}$. In the second application we set $D_a = D_4$, $D_b = D_5$, and $\{(v_i, u_i) : i = 1, \dots, \kappa\} = \{(z_i, \tilde{y}_i) : i = 1, \dots, \kappa\}$. Now, the expander D_a is used to connect the v_i's to the u_i's with *short* paths one at a time. In order to be sure that such paths exist we must be working with an expander. Therefore we delete some vertices in the course of the algorithm; in other words, this expander shrinks as the algorithm progresses. D_b is used to keep as many vertices as possible connected to the "shrinking expander" contained in D_a. We should note that these connections also require that D_b be an expander. So, D_b also "shrinks" in the course of the algorithm. The subroutines Remove and ConnectBack are used by GenPaths.

2.2.1. Initialization. Let $\tilde{X}, \tilde{Y}$ be two randomly chosen κ-subsets of V. We begin by replacing the problem of finding paths from x_i to y_i by that of finding paths from a_i to b_i, where $a_i \in \tilde{X}$ and $b_i \in \tilde{Y}$. Let X denote the set $\{x_1, x_2, \dots, x_\kappa\}$ and $Y = \{y_1, y_2, \dots, y_\kappa\}$. We connect X to $\tilde{X}$ via arc-disjoint paths in the digraph D_1 using a network flow. We construct our network as follows:

- Each directed arc of D_1 gets capacity 1.
- Each $v \in V$ becomes a source of capacity $|\{i : x_i = v\}|$, and each member of $\tilde{X}$ becomes a sink of capacity 1.

Then we find a flow from X to $\tilde{X}$ that satisfies all demands. We can find such a maximum flow with integer values, and this decomposes naturally into $|X|$ arc-disjoint

paths (together perhaps with some cycles). We connect $\tilde{Y}$ to Y by arc-disjoint paths in a similar manner using D_6

We now have a collection of arc-disjoint paths $\mathcal{P}^1 = \{P_i^1 : i = 1, \ldots, \kappa\}$ such that the starting point of P_i^1 is x_i for $i = 1, \ldots, \kappa$ and each member of $\tilde{X}$ is the endpoint of exactly one path from $\mathcal{P}^1$. Furthermore, we have a collection of arc-disjoint paths $\mathcal{P}^6 = \{P_i^6 : i = 1, \ldots, \kappa\}$ such that each member of $\tilde{Y}$ is the starting point of exactly one path from $\mathcal{P}^6$ and the endpoint of path P_i^6 is y_i for $i = 1, \ldots, \kappa$.

2.2.2. Algorithm GenPaths. The aim of GenPaths is to join v_i and u_i for $i = 1, 2, \ldots, \kappa$ by a *short* (i.e., of length $O(\log n)$) path in D_a. After constructing a path, we remove its arcs. It is important to ensure that short paths exist. Of course, this would not be a problem if we could ensure that D_a remained an expander throughout. We have to be satisfied with identifying a *dynamically changing large* subgraph $\Delta_a = (V_a, F_a)$ of D_a which is an expander. Initially $\Delta_a = D_a$, and V_a loses vertices as GenPaths progresses. We ensure that Δ_a remains an expander by keeping the degrees of vertices in the Δ_a close to their degree in D_a. This may involve deleting some (low degree) vertices after the construction of a path. We use the routine Remove to do this.

If the proposed start vertex v of a walk on Δ_a does not lie in V_a, then we try to *connect it back* to V_a by a path in D_b. The terminal endpoint of this walk is denoted by v'. We use a subroutine ConnectBack for this purpose. Similarly, the proposed end vertex u might not lie in V_a. In this case we use ConnectBack to find a path from some $u' \in V_a$ to u in D_b. We do not expect to succeed all the time and our failures are kept in a set L for treatment in Phase 2. The arcs in the paths generated by ConnectBack are deleted from D_b. Since ConnectBack requires that D_b is an expander we will also be working with a second "shrinking expander" $\Delta_b = (V_b, F_b)$ contained in D_b. This shrinking expander will also be maintained by use of the subroutine Remove.

In the end, the path from v_i to u_i will be a concatenation of up to three separate paths. There will always be a path Q_i from D_a, and there may also be a short walk (or walks) from D_b provided by ConnectBack (these are denoted $W_i^{CB\rightarrow}$ and $W_i^{CB\leftarrow}$, respectively).

1. **Algorithm** GenPaths
2. **begin**
3. $\Delta_t \leftarrow D_t$, $t = a, b$.
4. **for** $i = 1$ **to** κ **do**
5. **Execute** Remove(Δ_a)
6. **Execute** ConnectBack($V_a, v_i, \rightarrow, v_i', i, W_i^{CB\rightarrow}$)
7. **Execute** ConnectBack($V_a, u_i, \leftarrow, u_i', i, W_i^{CB\leftarrow}$)
8. **if** $i \notin L$ **then**
9. Construct a shortest path Q_i from v_i' to u_i' in Δ_a.
10. $P_i \leftarrow (W_i^{CB\rightarrow}, Q_i, W_i^{CB\leftarrow})$
11. $\Delta_a \leftarrow \Delta_a \setminus E(P_i)$
12. **fi**
13. **od**
14. **end** GenPaths

2.2.3. Subroutine Remove. The purpose of Remove is to delete vertices that might prevent a digraph from being an expander. In the course of GenPaths we apply REMOVE to Δ_a and Δ_b. In other words, this simple algorithm iteratively removes those vertices whose in-/out-degree is less than the original in-/out-degree

minus $\beta_0 r/2$. To be precise, at the end of the algorithm we have a graph $\Delta_t = (V_t, F_t)$, where $t \in \{a, b\}$ such that

$$v \in V_t \quad \text{implies} \quad d^\star_{\Delta_t}(v) \geq d^\star_{D_t}(v) - \beta_0 r/2 \geq \beta_0 r/2. \tag{7}$$

The final inequality in (7) follows from (6). It follows immediately from (7) that for $S \subseteq V_t$ we have

$$d^\star_{\Delta_t}(S) \geq d^\star_{D_t}(S) - \beta_0 r|S|/2 \geq (\Phi_t - \beta_0 r/2)|S|.$$

This implies that, provided neither V_a nor V_b become empty (an issue which we take up in the next section), both Δ_a and Δ_b are expanders throughout Phase 1:

$$\Phi_{\Delta_t} \geq \Phi_t - \beta_0 r/2 \geq \beta_0 r/2 \qquad \text{for } t = a, b. \tag{8}$$

1. **Algorithm** REMOVE(Δ_t)
2. **begin**
3. $B \leftarrow \{v \in V_t : \ d^+_{\Delta_t}(v) < d^+_{D_t}(v) - \beta_0 r/2 \ \text{ or } \ d^-_{\Delta_t}(v) < d^-_{D_t}(v) - \beta_0 r/2\}$.
4. **if** $B \neq \emptyset$ **then**
5. $A \leftarrow V_t \setminus B$
6. $d \leftarrow \max\{ \max\{d^+_{D_t}(v) - d^+_{\Delta_t}(v, A) \ , \ d^-_{D_t}(v) - d^-_{\Delta_t}(v, A)\} : \ v \in A\}$.
7. **while** $d > \beta_0 r/2$ **do**
8. $C \leftarrow \{w \in A : \max\{d^+_{D_t}(w) - d^+_{\Delta_t}(w, A) \ , \ d^-_{D_t}(w) - d^-_{\Delta_t}(w, A)\} \ \geq \beta_0 r/2\}$
9. $B \leftarrow B \cup C$
10. $A \leftarrow A \setminus C$
11. $d \leftarrow \max\{ \max\{d^+_{D_t}(v) - d^+_{\Delta_t}(v, A) \ , \ d^-_{D_t}(v) - d^-_{\Delta_t}(v, A)\} : \ v \in A\}$.
12. **od**
13. $V_t \leftarrow A$
14. **fi**
15. **end** REMOVE

2.2.4. Subroutine ConnectBack. The purpose of ConnectBack is to connect a vertex z to V_a by means of a short walk in D_b. The direction of this walk is determined by the input dir. If dir $= \rightarrow$, then a path from z to V_a is required, and if dir $= \leftarrow$, then a path from V_a to z is needed. If $z \in V_a$ already, then ConnectBack does nothing but relabel z as z'. Since $|V \setminus V_a|$ can be of order n (this is discussed below), we must maintain the expander property of Δ_b in order to find short connecting paths. Thus we apply Remove to Δ_b in the course of ConnectBack. Now, those pairs that contain a vertex that lies in $V \setminus (V_a \cup V_b)$ are passed to Phase 2 in the set L. Thus, the long-term success of ConnectBack hinges on keeping $V \setminus (V_a \cup V_b)$ small. In fact, this can be viewed as the key point in all of GenPaths.

We keep V_b large by ensuring that the paths we use in Δ_b are spread out; in other words, we avoid using too many paths through any one vertex. This is achieved **whp**; recall that the pairs of vertices $\{(v_i, u_i) : i = 1, \ldots, \kappa\}$ that are the input to GenPaths are assumed to be generated uniformly at random. When a path is needed (i.e., when $z \notin V_a$) ConnectBack constructs a collection $\mathcal{W}^{\text{dir}}$ of walks in Δ_b. This collection has the following properties:

1. If dir $= \rightarrow$, then every walk in $\mathcal{W}^{\text{dir}} = \mathcal{W}^{\rightarrow}$ is a walk from a distinct vertex in $V_b \setminus V_a$ to V_a.
2. If dir $= \leftarrow$, then every walk in $\mathcal{W}^{\text{dir}} = \mathcal{W}^{\leftarrow}$ is a walk from V_a to a distinct vertex in $V_b \setminus V_a$.

3. No path in $\mathcal{W}^{\text{dir}}$ is longer than $22 \log\log n$.
4. No vertex of D_b lies on more than $240(\log\log n)^2$ paths.

The set of start vertices of the walks in $\mathcal{W}^{\rightarrow}$ is denoted $S^{\rightarrow}_{CB}$, and the set of terminal vertices of the walks in $\mathcal{W}^{\leftarrow}$ is denoted $S^{\leftarrow}_{CB}$. Clearly, $S^{\text{dir}}_{CB} \subseteq V_b \setminus V_a$. The collection of walks will have the following additional property:

5. $|V \setminus (V_a \cup S^{\text{dir}}_{CB})| \leq \frac{n}{(\ln n)^4}$.

In other words, condition 3 says that the paths in $\mathcal{W}^{\text{dir}}$ are short, condition 4 says that the paths are "spread out" and condition 5 says that very few vertices are left out of the collection.

We emphasize that $\mathcal{W}^{\text{dir}}$ is constructed without use of any information about z. Therefore, z (which was a random vertex to begin with) can be viewed as a vertex chosen uniformly at random *after* the collection $\mathcal{W}^{\text{dir}}$ is constructed. Heuristically, we can think of $\mathcal{W}^{\rightarrow}$ and $\mathcal{W}^{\leftarrow}$ as collections of connecting paths that are updated whenever Δ_a or Δ_b "shrink," but we only "look" at these collections when we need them.

If $z \in S^{\text{dir}}_{CB}$, then we connect z back to V_a by way of the unique path in $\mathcal{W}^{\text{dir}}$that begins at z (if dir $= \rightarrow$) or ends at z (if dir $= \leftarrow$). If z does not lie in S^{dir}_{CB}, we put i into L (note that we have either $z = \tilde{x}_i$ or $z = \tilde{y}_i$). Arc-disjoint paths for the pairs $(\tilde{x}_i, \tilde{y}_i), i \in L$ are found in Phase 2.

A network-flow technique for the construction of the collection of paths $\mathcal{W}^{\text{dir}}$ follows from the proof of Lemma 3, which is given in section 2.3.3.

```
1.  subroutine ConnectBack(V_a, z, dir, z', i, W_CB)
2.  begin
3.     if z ∈ V_a
4.        then z' ← z
5.        else
6.           Execute Remove(Δ_b)
7.           Construct W^dir (see Lemma 3 for algorithm).
8.           if z ∉ S^dir_CB
9.              then L ← L ∪ {i}
10.             else
11.                W_CB ← the unique path in W^dir with start/terminal vertex z
12.                z' ← terminal/start vertex of W_CB
13.                Δ_b ← Δ_b \ W_CB
14.          fi
15.    fi
16. end ConnectBack
```

2.3. Analysis of Phase 1. There are three facts concerning Phase 1 that remain to be shown: that the flow needed in the initialization exists, that V_3 stays large, and that at the end of Phase 1 **whp** we have $L = O(\frac{n}{(\log n)^4})$.

2.3.1. Initialization. In this subsection we show that if (5) holds and r is sufficiently large, then we can find arc-disjoint paths from $\{x_1, \ldots, x_\kappa\}$ to $\tilde{X}$ in D_1 and arc-disjoint paths from $\tilde{Y}$ to $\{y_1, \ldots, y_\kappa\}$ in D_6 for *any* choice of $x_1, \ldots, y_\kappa$ consistent with the premises of Theorem 1, and for every choice for $\tilde{X}, \tilde{Y}$. We assume that we have

$$8\alpha > \epsilon_2 > r^{-1}. \tag{9}$$

For $S \subseteq V$, let

$$\alpha(S) = \sum_{v \in S} |\{i : x_i = v\}| \qquad \text{and} \qquad \xi(S) = |S \cap \tilde{X}|.$$

It follows from a theorem of Gale [12] (see Bondy and Murty [3, Theorem 11.8]) that if

$$d^+_{D_1}(S) \geq \xi(\bar{S}) - \alpha(\bar{S}) \quad \text{for all } S \subseteq V, \tag{10}$$

then there exists a flow in the network defined on D_1 such that exactly one unit of flow travels through each vertex in $\tilde{X}$, and the amount of flow traveling through each vertex $v \in \{x_1, \ldots, x_\kappa\}$ is $|\{i : x_i = v\}|$. In other words, (10) implies a successful run of Phase 2.

Now, if $|S| \leq n/2$, then, applying (4), (5), and (9), we have

$$d^+_{D_1}(S) \geq |S|\Phi_1 \geq |S|\beta_0 r \geq 8\alpha r|S| \geq \epsilon_2 r|S| \geq \alpha(S) - \xi(S) = \xi(\bar{S}) - \alpha(\bar{S}).$$

On the other hand, if $|S| > n/2$, then we have

$$d^+_{D_1}(S) = d^-_{D_1}(\bar{S}) \geq |\bar{S}|\Phi_1 \geq |\bar{S}|\beta_0 r \geq \epsilon_2 r|\bar{S}| \geq \xi(\bar{S}) - \alpha(\bar{S}).$$

Therefore, Phase 1 succeeds with respect to $X, \tilde{X}$. The same argument applies to $Y, \tilde{Y}$. To ensure these paths are of length $O(\log n)$ we can solve a minimum cost maximum flow problem as indicated in Kleinberg and Rubinfeld [13].

2.3.2. On the size of V_a.

LEMMA 2. *Throughout* GENPATHS *we have*

$$|V_a| \geq (1 - \gamma_0)n,$$

where

$$\gamma_0 = \frac{\beta_0 \gamma}{10}.$$

Proof. It follows from (8) that Δ_a is a $(\beta_0 r/2)$-expander throughout the execution of Phase 1. It follows from Lemma 1 that the diameter of Δ_a is always at most

$$\tau = \left\lceil 2\log_3 n + \frac{2}{\alpha\gamma} \right\rceil. \tag{11}$$

Indeed, consider breadth-first search from some $v \in V_a$. Let $L_t, t \geq 0$, be the vertices at distance t from v. Lemma 1 implies that the cardinalities of the L_t grow at a rate at least 3 until they reach size $(2\gamma\alpha/\beta_0)n$. The same will be true for breadth-first search to a target vertex w. This accounts for the first term in (11). Once L_t reaches $(2\gamma\alpha/\beta_0)n$, we use the fact that going to the next level involves finding $\beta_0 r|L_t|/2$ "new arcs," at least until size $n/2$ is reached. This accounts for the second term in (11).

Thus the total number of arcs in the paths that are removed from D_a is at most $\kappa\tau$. Let B be the set of vertices that are removed from Δ_a in the course of Phase 1, and let B_1 be the set of vertices in B incident with at least $\beta_0 r/4$ of the paths that are generated in D_a. We have

$$|B_1| \leq \frac{4\kappa\tau}{\beta_0 r} \leq \frac{\gamma_0 n}{2},$$

provided

$$\epsilon_1 \leq \frac{\beta_0^2 \gamma}{160}, \tag{12}$$

where ϵ_1 is as in the statement of Theorem 1. Let $B_2 = B \setminus B_1$ (i.e., those vertices removed from Δ_a that lie on less than $\beta_0 r/4$ of the paths generated in D_a).

Assume for the sake of contradiction that $|B_2| > |B_1|$. Let B_3 be the *first* $|B_1|$ vertices of B_2 to join B. Note that the vertices in B_3 have a large degree to $B_1 \cup B_3$ (otherwise these vertices would remain in V_a). Applying (4) we have

$$in_{D_a}(B_1 \cup B_3) \geq \frac{\beta_0 r}{4}|B_3| = \frac{\beta_0 r}{8}|B_1 \cup B_3| > \alpha r |B_1 \cup B_3|.$$

This contradicts (1).

Therefore, $|B| = |B_1| + |B_2| \leq 2|B_1| \leq \gamma_0 n$. □

2.3.3. Analysis of ConnectBack. Of course, the first order of business here is to show how the collection of paths $\mathcal{W}^{\mathrm{dir}}$ is generated.

Lemma 3. *Suppose that $D = (V, A)$ is an (α, β, γ)-expander and that $D' = (V', A')$ is a subdigraph of D of expansion at least θr where $\theta > 6\alpha$. Suppose that $S \subseteq V'$ and that $|S| \geq (1 - \frac{\gamma\alpha}{\theta})n$, and let $T = V' \setminus S$. Then there exists $T^\star \subseteq T$ such that D' contains a collection of walks $\mathcal{W}^{\mathrm{dir}} = \{W_v : v \in T^\star\}$ such that*

1. *for* $\mathrm{dir} = \rightarrow / \leftarrow$, *$v$ is the start/terminal vertex of W_v for all $v \in T^\star$;*
2. *the terminal/start vertex of each W_v is in S;*
3. *each W_v is of length at most $22 \log\log n$;*
4. *no vertex of D' lies on more than $240(\log\log n)^2$ paths; and*
5. *$|T \setminus T^\star| \leq \frac{n}{(\log n)^4}$.*

Proof. Assume without loss of generality that $\mathrm{dir} = \rightarrow$. For $i = 1, 2, \ldots,$ let

$$T_i = \{v \in T : \ dist_{D'}(v, S) = i\},$$

and set $T_0 = S$. Since $N^+_{D'}(\cup_{k \geq i} T_k) \subseteq T_{i-1}$, it follows from Lemma 1 that we have

$$|T_{i-1}| \geq \zeta |T_i| \qquad \text{for } i \geq 1, \tag{13}$$

where $\zeta = \frac{\theta - \alpha}{\alpha} > 5$. Setting $i_0 = \lceil 11 \log\log n \rceil$ and $\hat{T} = \bigcup_{i \geq i_0} T_i$, it follows from (13) that we have

$$|\hat{T}| \leq \frac{n}{2(\log n)^4}. \tag{14}$$

Fix $1 \leq i < i_0$. We define a flow network $\mathcal{N}_i$. The vertex set of $\mathcal{N}_i$ is $\{s, t\} \cup \bigcup_{j=1}^{j_0}(C_j \cup C'_j)$, where C_1 and C'_1 are disjoint copies of T_i for $2 \leq j \leq i$, C_j and C'_j are disjoint copies of $\bigcup_{\ell \geq i+1-j} T_\ell$ and, for $i < j \leq j_0 = 2i_0$, C_j and C'_j are disjoint copies of V'. The vertices s and t will be the source and sink, respectively, for the flow we introduce to $\mathcal{N}_i$. A vertex v in V' may appear many times in the vertex set of $\mathcal{N}_i$; a copy of v in C_j is denoted v_j, and a copy of v in C'_j is denoted v'_j. For ease of notation, we let ϕ be the map that takes the vertices of $\mathcal{N}_i$ to their corresponding vertices in V'. The arc-set of $\mathcal{N}_i$ is defined as follows. There is an arc from s to each vertex of C_1. Each $v \in S$ gives rise to arcs $(v'_j, t), i < j \leq j_0$. If $v'_j \in C'_j$ and $w_{j+1} \in C_{j+1}$ are such that (v, w) is an arc of D', then (v'_j, w_{j+1}) is an arc of $\mathcal{N}_i$. All arcs described so far have infinite capacity. In addition there are arcs (v_j, v'_j) of unit capacity defining a perfect matching between C_j and C'_j for $1 \leq j \leq j_0$.

CLAIM 1. *$\mathcal{N}_i$ contains an $s-t$ flow of value at least $|T_i| - \frac{n}{(\log n)^5}$.*

We first show the lemma follows from Claim 1. The flow given by Claim 1 defines paths in D' from all but at most $\frac{n}{(\log n)^5}$ vertices of T_i to S, each of length at most j_0. No vertex of V' can be on more than j_0 paths since each visit to v uses a (v_j, v'_j) arc for some j. Repeating this construction for $i = 1, 2, \dots, i_0$ we find paths for all but a set $\tilde{T}$ of at most $i_0 \frac{n}{(\log n)^5} \le \frac{n}{2(\log n)^4}$ vertices, and no vertex can be on more than $i_0 j_0$ paths. Putting $T^\star = T \setminus (\hat{T} \cup \tilde{T})$ and using (14) gives us the lemma.

It only remains to prove Claim 1. Of course, we do this via the max-flow min-cut theorem. Consider a cut $Z \dot\cup \bar{Z}$ of $\mathcal{N}_i$, where the vertex set Z contains s but not t. Let $A_j = Z \cap C_j$, $B_j = C_j \setminus A_j$, $A'_j = Z \cap C'_j$, and $B'_j = C'_j \setminus A'_j$ for $j = 1, 2, \dots, j_0$. Assume for the sake of contradiction that the capacity of this cut is less than $|C_1| - \frac{n}{(\log n)^5}$. It follows from this assumption that the cut contains no infinite capacity arcs and therefore

(15)
$$A_1 = C_1, \qquad \phi(A_{j+1}) \supseteq N^+_{D'}(\phi(A'_j)) \quad \text{for all } j, \qquad \phi(A'_j) \cap S = \emptyset \quad \text{for all } j.$$

The capacity of the cut is

$$|B'_1| + \sum_{j=2}^{j_0} |\phi(A_j) \cap \phi(B'_j)|. \tag{16}$$

The third condition of (15) implies that for all j, $|A'_j| < \frac{\gamma\alpha}{\theta} n$. It then follows from Lemma 1 and the second condition of (15) that we have $|A_{j+1}| \ge \zeta |A'_j|$. This implies that for all j we have either

$$|A'_{j+1}| \ge \frac{1}{2}|A_{j+1}| \ge \frac{\zeta}{2}|A'_j| \tag{17}$$

or

$$|\phi(A_{j+1}) \cap \phi(B'_{j+1})| \ge \frac{1}{2}|A_{j+1}| \ge \frac{\zeta}{2}|A'_j|. \tag{18}$$

Now, if $|A'_1| \le \frac{n}{(\log n)^5}$, then $|B'_1| \ge |C_1| - \frac{n}{(\log n)^5}$, which contradicts our initial assumption. On the other hand, if $|A'_1| > \frac{n}{(\log n)^5}$, then, since $(\frac{\zeta}{2})^{j_0 - 1} > (\log n)^5$, condition (17) cannot always hold. Let $j_1 \ge 1$ be the first j for which (18) holds. We have

$$|\phi(A_{j_1+1}) \cap \phi(B'_{j_1+1})| \ge (\zeta/2)^{j_1} |A'_1| \ge |A'_1|.$$

The capacity of the cut is at least $|B'_1| + |A'_1| = |C_1|$. This is a contradiction. □

To get the collection of paths needed for CONNECTBACK we apply Lemma 3 with $D' = \Delta_b, V' = V_b, S = V_a \cap V_b$. So, for example, we have $S^{\mathrm{dir}}_{CB} = T^\star$. It remains to show that **whp** we have $|L| \le \frac{n}{(\log n)^4}$.

Note that Lemma 3 can be applied only if $V_a \cap V_b$ is large. However, by applying the proof of Lemma 2 and the fact that the paths generated by CONNECTBACK are short (length at most $22 \log\log n$) we see that $|V_b| = n - O(n \log\log n / \log n)$ throughout, and this is sufficient. (Furthermore, the assumption that $\theta > 6\alpha$ in Lemma 3 is justified, since it follows from this observation that the expansion of Δ_b is always at least $3\beta_0 r/4$.) However, we shall see that V_b is **whp** larger than this.

This is where we use the fact that the paths in $\mathcal{W}^{\mathrm{dir}}$ are spread out (i.e., the fact that there are at most $240(\log\log n)^2$ paths in $\mathcal{W}^{\mathrm{dir}}$ through any one vertex).

It follows from this fact that the probability that an arbitrary vertex w is on the path $W_i^{CB\mathrm{dir}}$ is at most $240(\log\log n)^2/n$. It follows that we have

$$\begin{aligned}\mathbf{Pr}\left(\left|\left\{i : w \in W_i^{CB\rightarrow}\right\}\right| + \left|\left\{i : w \in W_i^{CB\leftarrow}\right\}\right| \geq 20\right)&\\ \leq \mathbf{Pr}(B(2\kappa, 240(\log\log n)^2/n) \geq 20)&\\ \leq \binom{2\kappa}{20}\left(\frac{240(\log\log n)^2}{n}\right)^{20}&\\ = o((\log n)^{-19}).&\end{aligned} \tag{19}$$

Let B be the set of vertices which are removed from Δ_b by applications of Remove. Let X_1 be the set of vertices in B that are on at least 20 of the paths $W_i^{CB\rightarrow}, W_i^{CB\leftarrow}$. Note that X_1 contains the set B_1 introduced in the proof of Lemma 2 (B_1 is the collection of vertices taken out of V_b by Remove that are on many of the paths). It follows from (19) and Markov's inequality that **whp** we have $|X_1| \leq n/(\log n)^{18}$. Now, B (which contains $V \setminus V_b$ at every step) consists of B_1 together with extra vertices deleted by Remove. In total this will be at most $2|B_1|$ vertices removed by the argument of Lemma 2, following (12). Thus, $|B| \leq 2|B_1| \leq 2|X_1|$. Therefore

$$|B| \leq \frac{n}{(\log n)^{18}} \tag{20}$$

whp.

Now, a failure (i.e., the index i joining the set L) can occur in one of two ways. On one hand we have a failure if either v_i or u_i does not lie in $V_a \cup V_b$, and on the other hand a failure results when $u_i \in V_b \setminus (V_a \cup S_{CB}^{\rightarrow})$ or $v_i \in V_b \setminus (V_a \cup S_{CB}^{\leftarrow})$. It follows from (20) that the total number of failures of the first type is **whp** at most $\frac{n}{(\log n)^{18}}$. For failures of the second type we note that $v_1, \ldots, v_\kappa$ form a random sequence of size $o(n)$. The probability that a particular vertex gives a failure of the second kind is at most

$$\frac{|S_{CB}^{\mathrm{dir}}|}{n} = O\left(\frac{1}{(\log n)^4}\right).$$

Applying the Chernoff bound for the tails of the binomial, we see that **whp** the total number of failures of the second kind is $O(n/(\log n)^4)$.

Remark 4. Suppose D has the following vertex expansion property for small sets: $S \subseteq V$, $|S| \leq \frac{\gamma}{r}n$ implies that $|N_D^\star(S)| \geq (1-\alpha)r|S|$. The algorithm of Theorem 4 can be modified to split D so that each subgraph D_i and small S satisfies $|N_{D_i}^\star(S)| \geq \frac{1-30\alpha}{13}r|S|$. Then the shortest paths in Δ_a will be of length $O(\log_r n)$ and the claim in Remark 2 will follow.

2.4. Phase 2. The set of pairs $\{(\tilde{x}_i, \tilde{y}_i) : i \in L\}$ have not yet been connected by paths. We have seen that **whp** the number of such pairs, $|L|$, is at most $O(n/(\log n)^4)$. These pairs are dealt with by the algorithm described below which uses digraphs D_7–D_{13}.

The heart of the algorithm is a randomized method (based on a multicommodity flow result of Leighton and Rao [16]) for connecting pairs of vertices with arc-disjoint paths which works **whp** when the collection of pairs is generated uniformly at random. So, as in Phase 1, some preliminary steps must be taken in order to reduce the problem

of connecting an arbitrary set of pairs of vertices with arc-disjoint paths to the problem of connecting a random collection of pairs with arc-disjoint paths. We proceed directly to a description of the algorithm. Let $m = |L|$ and $\lambda = \lceil \log n \rceil$.

We begin by "amplifying" each start vertex $\tilde{x}_i, i \in L$, and each end vertex $\tilde{y}_i, i \in L$, to a collection of λ vertices. This process occurs in steps (a) and (b).

(a) In this step, we choose a collection of vertices $w_j, 1 \le j \le 2m$, and a collection of sets of vertices $W_j, 1 \le j \le 2m$, such that for $1 \le j \le 2m$,
   (i) $w_j \in W_j$;
   (ii) $|W_j| = \lambda$;
   (iii) the sets W_j are pairwise disjoint; and
   (iv) D_9 contains an arborescence with vertex set W_j and root w_j; for $1 \le j \le m$ this arborescence is directed away from the root, and for $m+1 \le j \le 2m$ this arborescence is directed toward the root.

   Following [17], we find these arborescences by partitioning large arborescences of D_9. We begin with a rooted spanning arborescence T with the property that all arcs are directed away from the root. We generate $W_1, \dots, W_m$ greedily from D_9, removing an arborescence from T once it is used. Of course, this process will divide T into a number of components. However, since the maximum degree of D_9 is r, the number of components produced in this process is at most $r\lambda m = O(\frac{nr}{(\log n)^3})$. Since any tree having at least λ vertices contains a subtree having exactly λ vertices, we will always be able to find the needed arborescences. We then apply REMOVE to D_9 less the vertex set $\cup_{i=1}^m W_i$ to produce an expander D_9'. It follows from the proof of Lemma 2 that D_9' has $n - o(n)$ vertices. We repeat the process described above (this time using D_9' and an arborescense directed toward the roots) to produce $W_{m+1}, \dots, W_{2m}$.

(b) Let $S_X = \{\tilde{x}_i : i \in L\}$ and $S_Y = \{\tilde{y}_i : i \in L\}$ denote the sets of vertices that need to be joined. Use a network flow algorithm (analogous to what is given in the initialization step of Phase 1) in D_7 to connect in an arbitrary manner the vertices of S_X to $W_X = \{w_1, \dots, w_m\}$ by m arc-disjoint paths. Using the same network flow algorithm in D_8, connect in an arbitrary manner the vertices of $W_Y = \{w_{m+1}, \dots, w_{2m}\}$ to S_Y by m arc-disjoint paths. The expansion properties of D_7 and D_8 ensure that such paths always exist (as we saw in the initialization step of Phase 1).

Let $\hat{x}_k$ (resp., $\hat{y}_k$) denote the vertex in W_X that was connected to the endpoint $\tilde{x}_k$ (resp., $\tilde{y}_k$). Our problem is now to find arc-disjoint paths joining $\hat{x}_k$ to $\hat{y}_k$ for $1 \le k \le m$. If w_t has been renamed as $\hat{x}_k$ (resp., $\hat{y}_k$), then rename the elements of W_t as $\hat{x}_{k,\ell}$ (resp., $\hat{y}_{k,\ell}$), $1 \le \ell \le \lambda$.

(c) Choose $\xi_j, 1 \le j \le \lambda m$, and $\eta_j, 1 \le j \le \lambda m$, uniformly at random from V without replacement. Using a network-flow algorithm (as in (b)) connect $\{\hat{x}_{k,\ell} : 1 \le k \le m, 1 \le \ell \le \lambda\}$ to $\{\xi_j : 1 \le j \le \lambda m\}$ by arc-disjoint paths in D_{10}. Similarly, connect $\{\eta_j : 1 \le j \le \lambda m\}$ to $\{\hat{y}_{k,\ell} : 1 \le k \le m, 1 \le \ell \le \lambda\}$ by arc-disjoint paths in D_{13}. Rename the other endpoint of the path starting at $\hat{x}_{k,\ell}$ (resp., ending at $\hat{y}_{k,\ell}$) as $x^*_{k,\ell}$ (resp., $y^*_{k,\ell}$).

(d) Choose $z^*_{k,\ell}, 1 \le k \le m, 1 \le \ell \le \lambda$, uniformly at random from V with replacement. (We sample without replacement in (c) to ensure that all vertices have demands 0 or 1 in the flow algorithm. Here it is convenient to sample with replacement so that these choices are independent.) Now, it is important to note that the pairs $x^*_{k,\ell}, z^*_{k,\ell}$ and the pairs $z^*_{k,\ell}, y^*_{k,\ell}$ are (when viewed

separately) perfectly random. We are using the same "trick" that we used in Phase 1 for replacing the problem of connecting arbitrary pairs to the problem of connecting random pairs.

The paths between pairs of the form $x^*_{k,\ell}, z^*_{k,\ell}$ and between pairs of the form $z^*_{k,\ell}, y^*_{k,\ell}$ are generated at random. Using the multicommodity flow algorithm of Leighton and Rao [16] find a collection of paths $P_{u,v;\theta}, u \neq v \in V, 1 \leq \theta \leq \nu_{u,v}$, where each $P_{u,v;\theta}$ is a path in D_{11} from u to v. These are the flow paths given by Theorem 18 in [16] ($\nu_{u,v}$ is simply the number of flow paths we have for the pairs u, v). Let $\mathcal{P}_{u,v} = \{P_{u,v;\theta} : 1 \leq \theta \leq \nu_{u,v}\}$. For each $P_{u,v;\theta}$ we will have a flow value $f_{u,v;\theta} > 0$, and we let $F_{u,v} = \sum_{\theta=1}^{\nu_{u,v}} f_{u,v;\theta}$. Theorem 18 promises the following:

(P1) For all arcs e of D_{11},

$$\sum_{(u,v,\theta):\, e \in P_{u,v;\theta}} f_{u,v;\theta} \leq 1.$$

(P2)

$$F_{u,v} \geq \frac{c_2}{n \log n}$$

for some absolute constant $c_2 > 0$.

(P3) The length of each path $P_{u,v;\theta}$ is at most $\lambda_1 = c_1 \log_r n$ for some absolute constant $c_1 > 0$.

For $u, v \in V$, let $P_{u,v}$ be the probability distribution over $\mathcal{P}_{u,v}$ where $P_{u,v}(P_{u,v;\theta}) = f(u,v,\theta)/F(u,v)$. Then for each k, ℓ choose $W'_{k,\ell}$ randomly from $\mathcal{P}_{x^*_{k,\ell}, z^*_{k,\ell}}$ using the distribution $P_{x^*_{k,\ell}, z^*_{k,\ell}}$ to select the path.

Let B'_k denote the *bundle* of paths $\{W'_{k,\ell}, 1 \leq \ell \leq \lambda\}$. Carry out the same construction in D_{12} and construct a bundle of paths $B''_k = \{W''_{k,\ell}, 1 \leq \ell \leq \lambda\}$, where $W''_{k,\ell}$ is a path from $z^*_{k,\ell}$ to $y^*_{k,\ell}$.

Let $\pi_0 = \max_e \mathbf{Pr}(e \in P)$, where P is a path chosen by (i) randomly choosing endpoints u, v and then (ii) choosing $P \in \mathcal{P}_{u,v}$ according to the distribution $P_{u,v}$. We have

$$\pi_0 = \max_e \sum_{P_{u,v;\theta} \ni e} \frac{1}{n^2} \cdot \frac{f(u,v,\theta)}{F(u,v)} \leq \frac{1}{n^2} \frac{n \log n}{c_2} = \frac{\log n}{c_2 n}.$$

We say that $W'_{k,\ell}$ is *bad* if there exists $k' \neq k$ such that $W'_{k,\ell}$ shares an arc with a walk in a bundle $B'_{k'}$.

Now, suppose the bundles in the set $\{B_j : j \neq k\}$ are *fixed*. The collection of paths involved in these bundles gives at most $m\lambda\lambda_1$ arcs. Thus, the probability that $W_{k,l}$ is bad, conditioning on what happens outside the bundle B_k, is at most

$$\pi_0 m \lambda \lambda_1 = O\left(\frac{1}{\log n}\right).$$

We say that index k is bad if either B'_k or B''_k contains more than $\lambda/3$ bad walks. If index k is not bad, then we can find a walk from $x^*_{k,\ell}$ to $y^*_{k,\ell}$ through $x^*_{k,\ell}$ for some ℓ which is arc-disjoint from all other walks. This gives a walk

$$x_k - \tilde{x}_k - \hat{x}_k - \hat{x}_{k,\ell} - x^*_{k,\ell} - z^*_{k,\ell} - y^*_{k,\ell} - \hat{y}_{k,\ell} - \hat{y}_k - \tilde{y}_k - y_k,$$

which is arc-disjoint from all other such walks.
The probability that index k is bad is at most

$$2\,\mathbf{Pr}(B(\lambda, O((\log n)^{-1})) \geq \lambda/3) = O(n^{-2}).$$

So with probability 1-o(1) there are no bad indices. □

Appendix A. Splitting an expander digraph. We prove two results on splitting D into $D_1 \cup \cdots \cup D_k$, where $D_i = (V, A_i)$. The first is nonconstructive and shows what might be achieved. The second is constructive and uses the first. The split produced by the second is not as good as that indicated by the first result. We use a subscript i to denote graph-theoretic constructs related to D_i. Thus $d_i^+(v)$ is the out-degree of v in D_i. Left unsubscripted, such things refer to D. Thus $d^-(v) = r$.

In section B we prove the following.

THEOREM 3. *Let $k \geq 2$ be a positive integer and let $\epsilon > 0$ be a small positive real number. Suppose that the r-regular digraph $D = (V, A)$ has edge expansion Φ and that we have*

$$\frac{r}{\ln r} \geq 7k\epsilon^{-2} \qquad \text{and} \qquad \Phi \geq 5\epsilon^{-2}k \ln 2er.$$

Then there exists a partition $A = A_1 \cup A_2 \cup \cdots \cup A_k$ such that for $1 \leq i \leq k$

$$\Phi_i \geq (1-\epsilon)\frac{\Phi}{k} \qquad \text{and} \qquad (1-\epsilon)\frac{r}{k} \leq \delta^*(D_i) \leq \Delta^*(D_i) \leq (1+\epsilon)\frac{r}{k}.$$

We then use this in Appendix C in the proof of Theorem 4.

THEOREM 4. *Suppose that the conditions of Theorem 3 hold, and suppose further that D is an (α, β, γ)-expander. Then there is a randomized polynomial time algorithm (running time $O(n^2 \ln n \ln \delta^{-1})$) which with probability at least $1-\delta$ constructs $A_1, A_2, \ldots, A_k$ such that*

$$\Phi_i \geq (1-\epsilon)\frac{\Phi}{k} - (\alpha+\epsilon)\, r$$

for $i = 1, 2, \ldots, k$.

Note that this theorem is useful only if $\Phi \geq cr$ for some c satisfying $c \gg \alpha$. For random r-regular digraphs we can take γ to be a small constant and $\alpha = O(\gamma + \frac{1}{\sqrt{r}})$. Also note that there is not enough time to verify that the algorithm succeeds. Instead, we assume it has and repeat the split if we fail to find the required paths.

Appendix B. Existence result. We prove Theorem 3. We will use the general version of the Lovász local lemma. For each $a \in A$ we randomly choose an integer $i \in [k]$ and then place a in A_i. We must show that there is a positive probability of choosing a partition which satisfies the conditions of the theorem.

We begin with some definitions and preliminary observations. Let $G = (V, E)$ be the $2r$-regular (multi-)graph obtained by ignoring orientation in D. If $S \subseteq V$, then $G[S]$ is the subgraph of G induced by S. We say that S is *connected* if $G[S]$ is.

CLAIM 2. *For $v \in V$ there are at most $(2er)^{s-1}$ sets S such that* (i) *$v \in S$,* (ii) *$|S| = s$, and* (iii) *S is connected.*

Proof of Claim 2. The number of such sets is bounded by the number of distinct s-vertex trees which are rooted at v. This in turn is bounded by the number of distinct $2r$-ary rooted trees with s vertices. This is equal to $\binom{2rs}{s}/((2r-1)s+1)$; see Knuth [15]. □

In both this section and what follows will use the following Chernoff bounds for the tails of the binomial distribution $B(n,p)$:

$$\mathbf{Pr}(B(n,p) \geq (1+\epsilon)np) \leq e^{-\epsilon^2 np/3}, \tag{21}$$

$$\mathbf{Pr}(B(n,p) \leq (1-\epsilon)np) \leq e^{-\epsilon^2 np/2}, \tag{22}$$

where $0 \leq \epsilon \leq 1$.

For our application of the Lovász local lemma, we define the following *bad* events:

(a) For $v \in V$, $i \in [k]$, and $* \in \{+,-\}$, $A_{v,i,*} = A_{\{v\},i,*}$ is the event that

$$d_i^*(v) \notin [(1-\epsilon)r/k, (1+\epsilon)r/k].$$

(b) For $S \subseteq V$, $2 \leq |S| \leq n/2$, S connected, $i \in [k]$, and $* \in \{+,-\}$, $A_{S,i,*}$ is the event that

$$|d_i^*(S)| < (1-\epsilon)|d^*(S)|/k.$$

In showing that Φ_i is sufficiently large we can restrict our attention to S for which S is connected. Indeed, for $S \subseteq V$ let $C_1, C_2, \ldots, C_t$ be the components of $G[S]$. Then for $* \in \{+,-\}$,

$$\Phi_{S,i}^* \geq \min_{1 \leq s \leq t} \frac{d_i^*(C_s)}{|C_s|}.$$

Using the Chernoff bounds given above, we obtain

$$\mathbf{Pr}(A_{v,i,*}) \leq 2e^{-\epsilon^2 r/(3k)} \leq 2e^{-(7\ln r)/3} < \frac{1}{r^2}$$

and

$$\mathbf{Pr}(A_{S,i,*}) \leq \exp\left\{-\frac{\epsilon^2 d^*(S)}{2k}\right\} \leq e^{-2|S|\ln r} = \frac{1}{r^{2|S|}}.$$

Now, for $S \subseteq V$, $1 \leq |S| \leq n/2$, and S connected, let

$$x_{S,i,*} = \left(\frac{2}{r^2}\right)^{|S|}.$$

We show that for $*, \# \in \{+,-\}$,

$$\mathbf{Pr}(A_{S,i,*}) < x_{S,i,*} \prod_{(S,i,*) \sim (T,j,\#)} (1 - x_{T,j,\#}), \tag{23}$$

where $(S,i,*) \sim (T,j,\#)$ denotes adjacency of $A_{S,i,*}$ and $A_{T,j,\#}$ in the dependency graph of bad events (i.e., we have $(S,i,*) \sim (T,j,\#)$ if and only if $A^*(S) \cap A^{\#}(T) \neq \emptyset$). The theorem then follows from the general version of the local lemma; see, for example, Alon and Spencer [1].

It follows from Claim 2 that if $|S| = s$, then there are at most $ks(2er)^t$ events $A_{T,j,\#}$ with $|T| = t$ such that $(S,i,*) \sim (T,j,\#)$. Thus, using $1 - x \geq e^{-2x}$ for

$0 \le x \le 1/2$, we have

$$\begin{aligned} x_{S,i,*} \prod_{(S,i,*)\sim(T,j,\#)} (1 - x_{T,j,\#}) &\ge \left(\frac{2}{r^2}\right)^s \prod_{t\ge 1} \left(1 - \left(\frac{2}{r^2}\right)^t\right)^{ks(2er)^t} \\ &\ge \left(\frac{2}{r^2}\right)^s \exp\left\{-2ks \sum_{t\ge 1} \left(\frac{4e}{r}\right)^t\right\} \\ &= \left(\frac{2}{r^2}\right)^s \exp\left\{-\frac{8kes}{r-4e}\right\} \\ &> \frac{1}{r^{2s}}, \end{aligned}$$

since for small values of ϵ, the fact that $r/\ln r \ge 7k\epsilon^{-2}$ implies

$$r > 4e + \frac{8ke}{\ln 2}.$$

Thus (23) holds, proving the theorem. □

Appendix C. Splitting algorithm. In this section, we prove Theorem 4.

Idea. We produce the split in a series of rounds that gives a series of vertex sets $V = B_1 \supseteq B_2 \supseteq \cdots \supseteq B_t$. In round i we fix the "destination" of a fixed arc if it has at least one endpoint in B_i but no endpoint in B_{i+1}. This is done in such a way that if $S \subseteq B_j \setminus B_{j+1}$, then $d_i^+(S), d_i^-(S)$ are large enough, and further that every vertex in $B_j \setminus B_{j+1}$ has few neighbors in B_{j+1}. We will see that this latter condition accounts for the $-(\alpha + 2\epsilon)\, r$ term in the theorem.

Assume that we have $B \subseteq V$. Initially, $B = V$. We randomly color the arcs of D *which are incident with* B with k colors. Note that if $s_0 = 5k\epsilon^{-2}\Phi^{-1}\ln n$,

$$\begin{aligned} &\mathbf{Pr}\left(\exists S \subseteq B, i \in [k] \text{ s.t. } |S| > s_0, S \text{ is connected and } \Phi_{i,S} \le (1-\epsilon)\,\frac{\Phi}{k}\right) \\ &\qquad \le 2kn \sum_{s\ge s_0} (2er)^{s-1} e^{-\epsilon^2 s\Phi/(2k)} \le 4kn(2er)^{s_0} e^{-\epsilon^2 s_0\Phi/(2k)} = O(kn^{-1/5}). \end{aligned}$$

So, in a sense the large sets take care of themselves. Now consider the smaller sets. Let

$$X_0 = \left\{v : \exists S \subseteq B, |S| \le s_0, S \text{ is connected}, v \in S \text{ and } i \in [k] \text{ s.t. } \Phi_{i,S} \le (1-\epsilon)\,\frac{\Phi}{k}\right\}.$$

X_0 can be constructed in $O(n(er)^{s_0}) = O(n^2)$ time.

$$\mathbf{E}(|X_0|) \le |B| \sum_{s=1}^{s_0} (2er)^{s-1} e^{-\epsilon^2 s\Phi/(2k)} \le \frac{|B|}{2er}$$

since $\Phi \ge 5\epsilon^{-2}k\ln 2er$.

Therefore by Markov's inequality,

$$\mathbf{Pr}\left(|X_0| \ge \frac{|B|}{er}\right) \le \frac{1}{2}.$$

We repeat the above coloring until we find that $|X_0| \le \frac{|B|}{er}$. Now recursively define $X_j = X_{j-1} \cup \{v_j\}$, where $d^+(v_j, X_{j-1}) \ge (\alpha+\epsilon)r$ or $d^-(v_j, X_{j-1}) \ge (\alpha+\epsilon)r$ if such a v_j exists. Here we use the strong expansion properties of an (α, β, γ)-expander: $|S| \le \gamma n$ implies that S contains at most

$$(r|S| - d^*(S)) \le \alpha r|S|$$

arcs. Note that X_j has at least $(\alpha+\epsilon)rj$ arcs and at most $j + \frac{|B|}{er}$ vertices. Thus this process stops before j reaches $\frac{\alpha|B|}{\epsilon er}$, unless $|X_j|$ exceeds γn first. However, this latter possibility cannot happen because $|X_0| + \frac{\alpha|B|}{\epsilon er} \le (1+\frac{\alpha}{\epsilon})\frac{1}{er}|B| \le \gamma|B| \le \gamma n$, since $\gamma, \epsilon > r^{-\frac{1}{2}}, \alpha < 1$ implies $\gamma > (1+\frac{\alpha}{\epsilon})\frac{1}{er}$.

So if X denotes X_j when v_{j+1} cannot be found, then

$$|X| \le \gamma|B|.$$

We will repeat the construction with B replaced by X. Let $V = B_1 \supseteq B_2 \supseteq \cdots \supseteq B_t$ be the sequence of sets constructed. B_t will be the first set of size at most $r^{-1}\ln n$. Since $\gamma < \frac{1}{2}$, we have $t \le \log_2 n$. Thus the expected number of recolorings needed is at most $2\log_2 n$ and is $\le 3\log_2 n$ **whp**. We can "brute force" color the arcs incident with B_t so that every subset S of B_t satisfies $\Phi_{i,S} \ge (1-\epsilon)\frac{\Phi}{k}$. We use Theorem 3 to justify the success of this. The sequence of sets $B_1, B_2, \ldots, B_t$ satisfies the following:

- $|B_j| \le \gamma^j n$;
- $S \subseteq B_j \setminus B_{j+1}$ implies $\Phi_{i,S} \ge (1-\epsilon)\frac{\Phi}{k}$;
- $v \in B_j \setminus B_{j+1}$ implies v has at most $(\alpha+\epsilon)r$ out-neighbors and at most $(\alpha+\epsilon)r$ in-neighbors in B_{j+1}.

So if $S \subseteq V$ and $S_j = S \cap (B_j \setminus B_{j+1})$,

$$\begin{aligned} d_i^*(S) &\ge \sum_{j=1}^{t-1}(d_i^*(S_j) - d_i^*(S_j, B_{j+1})) + d_i^*(S_t) \\ &\ge \sum_{j=1}^{t-1}\left((1-\epsilon)\frac{\Phi}{k} - (\alpha+\epsilon)r\right)|S_j| + (1-\epsilon)\frac{\Phi}{k}|S_t| \\ &\ge \left((1-\epsilon)\frac{\Phi}{k} - (\alpha+\epsilon)\,r\right)|S|. \qquad \square \end{aligned}$$

REFERENCES

[1] N. Alon and J. H. Spencer, *The Probabilistic Method*, John Wiley, New York, 1992.

[2] J. Bang-jensen and G. Gutin, *Digraphs: Theory, Algorithms and Applications*, Springer-Verlag, London, 2001.

[3] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North–Holland, Amsterdam, 1976.

[4] A. Z. Broder, A. M. Frieze, and E. Upfal, *Existence and construction of edge-disjoint paths on expander graphs*, SIAM J. Comput., 23 (1994), pp. 976–989.

[5] A. Z. Broder, A. M. Frieze, and E. Upfal, *Existence and construction of edge low congestion paths on expander graphs*, Random Structures Algorithms, 14 (1999), pp. 87–109.

[6] S. Fortune, J. E. Hopcroft, and J. Wyllie, *The directed subgraph homeomorphism problem*, Theoret. Comput. Sci., 10 (1980), pp. 111–121.

[7] A. Frank, *Disjoint paths in rectilinear grids*, Combinatorica, 2 (1982), pp. 361–371.

[8] A. M. Frieze, *Disjoint Paths in Expander Graphs via Random Walks: A Short Survey*, in Proceedings of Random '98, Lecture Notes in Comput. Sci. 1518, Springer-Verlag, New York, 1998, pp. 1–14.

[9] A. M. Frieze, *Edge-disjoint paths in expander graphs*, SIAM J. Comput., 30 (2001), pp. 1790–1801.
[10] A. M. Frieze and M. Molloy, *Splitting an expander graph*, J. Algorithms, 33 (1999), pp. 166–172.
[11] A. M. Frieze and L. Zhao, *Optimal construction of edge-disjoint paths in random regular graphs*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1999, pp. 346–355.
[12] D. Gale, *A theorem on flows in networks*, Pacific J. Math., 7 (1957), pp. 1073–1082.
[13] J. Kleinberg and R. Rubinfeld, *Short paths in expander graphs*, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, IEEE Press, Piscataway, NJ, 1996, pp. 86–95.
[14] J. Kleinberg and E. Tardos, *Approximations for the disjoint paths problem in high diameter planar networks*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1995, pp. 26–35.
[15] D. E. Knuth, *The Art of Computer Programming, Vol.* 1*, Fundamental Algorithms*, Addison–Wesley, Reading, MA, 1968.
[16] T. Leighton and S. Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, J. ACM, 46 (1999), pp. 787–832.
[17] T. Leighton and S. Rao, *Circuit switching: A multicommodity flow based approach*, in Proceedings of a Workshop on Randomized Parallel Computing, 1996.
[18] T. Leighton, S. Rao, and A. Srinivasan, *Multi-commodity flow and circuit switching*, in Proceedings of the Hawaii International Conference on System Sciences, 1998.
[19] T. Leighton, S. Rao, and A. Srinivasan, *New algorithmic aspects of the local lemma with applications to routing and partitioning*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, PA, 1999, pp. 643–652.
[20] A. Lubotsky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica, 8 (1988), pp. 261–277.
[21] D. Peleg and E. Upfal, *Constructing disjoint paths on expander graphs*, Combinatorica, 9 (1989), pp. 289–313.
[22] N. Robertson and P. D. Seymour, *Graph minors.* XIII: *The disjoint paths problem*, J. Combin. Theory Ser. B, 63 (1995), pp. 65–110.
[23] A. Sinclair and M. Jerrum, *Approximate counting, uniform generation, and rapidly mixing Markov chains*, Inform. and Comput., 82 (1989), pp. 93–133.
[24] D. Wagner and K. Weihe, *A linear time algorithm for edge-disjoint paths in planar graphs*, Proceedings of the First European Symposium on Algorithms (ESA '93), in Lecture Notes in Comput. Sci. 726, Springer-Verlag, New York, 1992, pp. 384–395.
[25] X. Zhou, S. Tamura, and T. Nishizeki, *Finding edge-disjoint paths in partial k-trees*, Algorithmica, 26 (2000), pp. 3–30.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 345–370

THE PROBABLE VALUE OF THE LOVÁSZ–SCHRIJVER RELAXATIONS FOR MAXIMUM INDEPENDENT SET*

URIEL FEIGE† AND ROBERT KRAUTHGAMER‡

Abstract. Lovász and Schrijver [*SIAM J. Optim.*, 1 (1991), pp. 166–190] devised a lift-and-project method that produces a sequence of convex relaxations for the problem of finding in a graph an independent set (or a clique) of maximum size. Each relaxation in the sequence is tighter than the one before it, while the first relaxation is already at least as strong as the Lovász theta function [*IEEE Trans. Inform. Theory*, 25 (1979), pp. 1–7]. We show that on a random graph $G_{n,1/2}$, the value of the rth relaxation in the sequence is roughly $\sqrt{n/2^r}$, almost surely. It follows that for those relaxations known to be efficiently computable, namely, for $r = O(1)$, the value of the relaxation is comparable to the theta function. Furthermore, a perfectly tight relaxation is almost surely obtained only at the $r = \Theta(\log n)$ relaxation in the sequence.

Key words. stable set polytope, semidefinite relaxation, lift-and-project, random graph, clique

AMS subject classifications. 05C69, 05C80, 90C22, 90C27

PII. S009753970240118X

1. Introduction.

Let $G(V, E)$ be a graph on n vertices. An *independent set* (also known as a *stable set*) in G is a subset of vertices no two of which are connected by an edge. The *maximum independent set* problem requires one to find an independent set of maximum size in an input graph G. The *independence number* (also known as a *stability number*) of G, denoted $\alpha(G)$, is the maximum size of an independent set in G.

A *clique* in G is a subset of vertices every two of which are connected by an edge. The *maximum clique* problem requires one to find a clique of maximum size in an input graph G. The *clique number* of G, denoted $\omega(G)$, is the maximum size of a clique in G. A clique in G forms an independent set in the edge complement graph $\overline{G}$, so $\omega(G) = \alpha(\overline{G})$. It follows that the maximum clique problem and the maximum independent set problem are equivalent in many respects, including the context presented here. For consistency with related literature, we refer to one problem in some parts and to the other problem in others.

The maximum independent set problem is fundamental in the area of combinatorial optimization and is closely related, in addition to the maximum clique problem, to the *vertex cover* problem (the vertex complement of an independent set) and the *chromatic number* problem (minimum cover by independent sets). The maximum independent set problem (or even finding $\alpha(G)$) is one of the first problems shown to be NP-hard in [16].

A common way to cope with the NP-hardness of a problem is to devise algorithms

*Received by the editors January 21, 2002; accepted for publication (in revised form) August 9, 2002; published electronically January 17, 2003.
http://www.siam.org/journals/sicomp/32-2/40118.html

†Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. Incumbent of the Joseph and Celia Reskin Career Development Chair (feige@wisdom.weizmann.ac.il).

‡Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel (robi@wisdom.weizmann.ac.il, robi@cs.berkeley.edu). The work of this author was supported in part by a Dora Ostre memorial scholarship. Currently at the International Computer Science Institute (ICSI) and the Computer Science Division, University of California, Berkeley, CA.

that give approximate solutions. An *efficient* (i.e., polynomial time) algorithm is said to have an *approximation ratio* $\rho > 1$ for the maximum independent set problem if for every input graph, the ratio between $\alpha(G)$ and the size of the independent set returned by the algorithm is at most $\rho = \rho(n)$. It is known through work culminating in [12] that for any fixed $\epsilon > 0$, it is impossible to approximate the independence number $\alpha(G)$ within a ratio of $n^{1-\epsilon}$, unless NP has randomized polynomial time algorithms (NP = ZPP). The best approximation algorithm that is known for $\alpha(G)$, due to [4], has approximation ratio $O(n/\log^2 n)$.

The intractability of the maximum independent set problem in the worst case suggests studying the performance of algorithms on average instances. A possible rigorous description of average instances is by probabilistic models; see, e.g., [8] for a survey on average-case analysis of graph algorithms on random graphs.

The problem of finding a maximum independent set on a random graph appears to be difficult. Let $G_{n,1/2}$ denote the random graph on n labeled vertices obtained by connecting each pair of vertices by an edge independently with probability 1/2. It is known that the independence number of $G_{n,1/2}$ is roughly $2\log_2 n$, almost surely, i.e., with probability that approaches 1 as n tends to infinity; see, e.g., [3]. Several simple and natural algorithms (e.g., the greedy one) find an independent set of size roughly $\log_2 n$, almost surely. However, no algorithm is known to find efficiently an independent set of size significantly larger than $\log_2 n$; see, e.g., [17, 8]. Finding independent sets of size $\frac{3}{2}\log_2 n$ in random graphs was even suggested as a hard computational problem on which to base cryptographic applications; see [14].

Lovász theta function. A well-known relaxation of the maximum independent set problem is the *theta* function of a graph, denoted $\vartheta(G)$, introduced by Lovász [21] (see also [11, Chapter 9] and Knuth's survey [18]). The theta function can be formulated as a semidefinite program and thus can be computed, up to arbitrary precision, in polynomial time; see, e.g., [11]. We may consider the theta function also as a relaxation of the maximum clique problem by formally referring to $\vartheta(\bar{G})$.

In terms of approximation ratio, the theta function appears to have little to offer. The ratio between $\vartheta(G)$ and the independence number $\alpha(G)$ can be as large as $n^{1-o(1)}$, as shown in [6].

Also, on the average there is a large gap between the Lovász theta function $\vartheta(G)$ and the independence number $\alpha(G)$. While the independence number of a random graph $G_{n,1/2}$ is almost surely roughly $2\log_2 n$, it was shown by Juhász [15] that the value of the theta function is almost surely $\Theta(\sqrt{n})$.

The hidden clique problem. Jerrum [13] and Kučera [20] suggested independently the following *hidden clique problem*. A random graph $G_{n,1/2}$ is chosen and a clique of size k is randomly placed in the graph. We wish to find in this graph a maximum clique. Jerrum showed that the Metropolis process almost surely does not find the clique when $k = o(\sqrt{n})$. Kučera observed that when $k > c\sqrt{n \log n}$ for an appropriate constant c, the vertices of the planted clique would almost surely be the ones with the largest degrees in G, and hence it is easy to recognize them efficiently. Alon, Krivelevich, and Sudakov [1] showed an algorithm that almost surely finds the planted clique whenever $k \geq \Omega(\sqrt{n})$. Their algorithm is based on spectral properties of the graph, namely, it uses the eigenvector that corresponds to the second largest eigenvalue of the adjacency matrix of the graph. (See also [24].)

Feige and Krauthgamer [7] devised another algorithm that is based on the semidefinite programming relaxation provided by the Lovász theta function. Their algorithm works for the same planted clique size k as the algorithm of [1], but it has the ad-

vantage of being more robust; it works also in a semirandom model in which an adversary can remove edges that are outside the planted clique. Another advantage of their algorithm is that it certifies almost surely the optimality of its solution.

The approach of [7] was motivated by Juhász' result [15] that the theta function of a random graph $G_{n,1/2}$ is $\Theta(\sqrt{n})$, almost surely. It follows that the maximum clique relaxation $\vartheta(\bar{G})$ is also almost surely $\Theta(\sqrt{n})$ for a random graph $G_{n,1/2}$. When a clique of size $k \geq c\sqrt{n}$ for a sufficiently large constant $c > 0$ is planted in a random graph, the theta function (being a relaxation) must increase to at least k. Furthermore, it is plausible that such a noticeable increase in the theta function will allow us to find the planted clique. Indeed, it is shown in [7] that on the hidden clique graph $G_{n,1/2,k}$, the theta function almost surely gives exactly k, the planted clique size, in which case it allows us to find the planted clique (with some extra work). In contrast, when a clique of size $k = o(\sqrt{n})$ is planted in a random graph, the monotonicity properties of the theta function (see, e.g., [18, sections 18–19]) guarantee that its value can only increase, but not by more than k. It follows that on the hidden clique graph $G_{n,1/2,k}$, the value of the theta function is also almost surely $\Theta(\sqrt{n})$, and it is therefore possible that the planted clique has no noticeable effect on the theta function.

A possible direction for extending the approach of [7] to a planted clique of smaller size $k = o(\sqrt{n})$ is to use relaxations that are stronger than the Lovász theta function. In particular, it is desirable to find a relaxation whose value on a random graph $G_{n,1/2}$ is almost surely $o(\sqrt{n})$.

The general Lovász–Schrijver technique. Lovász and Schrijver [23] propose a general technique for obtaining stronger and stronger relaxations of 0-1 integer programming problems. Specifically, they devise several procedures called *matrix-cut operators* that produce from a convex (e.g., linear programming) relaxation $P \subseteq [0,1]^n$ of the problem a convex set that is an improved relaxation for the 0-1 (i.e., integral) vectors in P. That is, the resulting convex set is contained in P and contains all the 0-1 vectors in P. The matrix-cut operators follow a *lift-and-project* approach; they lift the convex relaxation P into a higher (quadratic) dimension by introducing new variables and new constraints, and then project it back into the original space.

The two main matrix-cut operators of Lovász and Schrijver [23] are denoted by N and N_+. The difference between the two operators is that the lifting of the latter involves, in addition, a positive semidefinite constraint. That is, if P is a linear programming relaxation, then $N(P)$ is also a linear programming relaxation, while $N_+(P)$ is a semidefinite programming relaxation.

The matrix-cut operators can be applied iteratively, say $r \geq 0$ times, and the iterated operators are denoted N^r and N_+^r. The *N-rank of a convex relaxation P* is defined as the number of iterations of the N operator that are needed to obtain the convex hull of the 0-1 vectors of P (i.e., a perfectly tight relaxation). The *N_+-rank* is defined similarly. Lovász and Schrijver [23] show that the N-rank of a relaxation is always at most the dimension d (e.g., number of variables in a linear program). The N_+ operator is a strengthening of the N operator, and hence the N_+-rank is also always at most d. Goemans and Tunçel [10] and Cook and Dash [5] show independently that there exist relaxations whose N_+-rank meets the upper bound d.

Furthermore, Lovász and Schrijver [23] show that the N and N_+ operators have the following important algorithmic property. If it is possible to efficiently optimize (linear objective functions) over a relaxation P, then it is also possible to efficiently optimize over the relaxation obtained by applying the operator on P. It follows that for every *fixed* $r \geq 0$, the iterated operators N^r and N_+^r also satisfy this property.

Strong relaxations for maximum independent set. To obtain relaxations of the maximum independent set problem, Lovász and Schrijver [23] apply their general technique of matrix-cut operators on a classical linear programming relaxation FRAC of the problem. The relaxation FRAC is a linear program of polynomial size, and hence for every fixed $r \geq 0$, one can efficiently optimize over $N_+^r(\text{FRAC})$. In contrast, the dimension d (i.e., number of variables) of FRAC is the number of vertices n in the graph, and so optimizing over $N^n(\text{FRAC})$ is NP-hard.

Lovász and Schrijver [23] show that the semidefinite programming relaxation $N_+(\text{FRAC})$ is at least as strong as the Lovász theta function. It follows, for example, that for any graph on which the theta function is not tight, the relaxation $N_+^r(\text{FRAC})$ for $r \geq 2$ is stronger than the theta function.

The *N-rank of a graph* is defined as the N-rank of the relaxation FRAC. The *N_+-rank* is defined similarly. It follows that for graphs with bounded N_+-rank, the maximum independent set problem can be solved in polynomial time. This family includes, for example, all perfect graphs, since the above connection with the theta function implies that their N_+-rank is at most 1.

Stephen and Tunçel [25] study the case where the n-vertex graph G is the line graph of a graph H on h vertices. They show that the N_+-rank of G is at most $\lfloor h/2 \rfloor$, and that this bound is met if H is a complete graph on an odd number of vertices, in which case $n = \binom{h}{2}$, and so the N_+-rank of G is $\Omega(\sqrt{n})$. Note that independent sets in G correspond to matchings in H, and that a maximum weight matching can be found efficiently; it follows that there are graphs with unbounded (and rather large) N_+-rank, in which the maximum (weighted) independent set problem can be solved in polynomial time.

Our results. We examine the asymptotic behavior on the random graph $G_{n,1/2}$ of the relaxations of Lovász and Schrijver [23] for the maximum independent set problem. In particular, we show that the typical value of the semidefinite programming relaxation $N_+^r(\text{FRAC})$ on a random graph is roughly $\sqrt{n/2^r}$ for $r = o(\log n)$. We note that this characterization answers (up to a constant factor) a question of Knuth [18, section 37, Problem P6].

THEOREM 1.1. *For every fixed $\delta > 0$ and $r = o(\log n)$, the value of the relaxation $N_+^r(\text{FRAC})$ on a random graph $G_{n,1/2}$ is at least $\sqrt{n/(2+\delta)^{r+1}}$ and at most $4\sqrt{n/(2-\delta)^{r+1}}$, almost surely.*

Recall that the strongest relaxations of Lovász and Schrijver [23] whose value is known to be efficiently computable are $N_+^r(\text{FRAC})$ for $r = O(1)$. Theorem 1.1 shows that on a random graph, the typical value of these relaxations is smaller than that of the theta function by no more than a constant factor. In the hidden clique problem, the planted clique size k that a heuristic can handle can be improved by an arbitrarily large constant factor using a method of [1], and therefore it appears that the improvement offered by these stronger relaxations can be achieved by other methods.

We use Theorem 1.1 to characterize, up to a constant factor, the typical N_+-rank of a random graph $G_{n,1/2}$.

THEOREM 1.2. *The N_+-rank of a random graph $G_{n,1/2}$ is almost surely $\Theta(\log n)$.*

Our results for the N_+ operator extend to a slightly stronger variant of the matrix-cut operators of Lovász and Schrijver [23]. This operator, denoted $N_{\text{FR}+}$, is specialized for the maximum independent set problem and retains the important algorithmic property of N_+, namely, an efficient optimization over P implies an efficient optimization over $N_{\text{FR}+}(P)$.

Organization. Section 2 is a technical description of the matrix-cut operators of Lovász and Schrijver [23] (including our variant $N_{\mathrm{FR}+}$). We present the formal definitions in section 2.1 and state in section 2.2 some basic useful properties (whose proof is deferred to Appendix A.1).

Section 3 describes our results on matrix-cuts in a random graph. Specifically, a lower bound on the value of the relaxation $N_+^r(\mathrm{FRAC})$ is shown in section 3.1 and an upper bound is shown in section 3.2.

The appendix proves several useful properties of the matrix-cut operators. In section A.1 we give some basic properties that are needed for our main results, and in section A.2 we give bounds on the ranks of the different matrix-cut operators. Most of the results on the N and N_+ operators were previously published in [23, 5, 10] and are included here for completeness. The results on the $N_{\mathrm{FR}+}$ operator were not published previously (to the best of our knowledge).

Preliminaries. Throughout, we omit the graph $G(V, E)$ if it is clear from the context. We let n denote the number of vertices in the graph G and assume, without loss of generality, that $V = \{1, \ldots, n\}$. For a vertex i in the graph, let $\Gamma(i)$ denote the set of the vertices that are adjacent to i in the graph, i.e., $\Gamma(i) := \{j \in V : ij \in E\}$, and let $\Gamma(S)$ denote the set of vertices in V that are adjacent to at least one vertex of S, i.e., $\Gamma(S) := \cup_{i \in S} \Gamma(i)$.

An $n \times n$ (real) matrix Y is *positive semidefinite* if Y is symmetric and $x^T Y x \geq 0$ for all $x \in \mathbb{R}^n$. It is well known that a symmetric matrix Y is positive semidefinite if and only if all the eigenvalues of Y are nonnegative.

A *Gram matrix representation* of an $n \times n$ matrix Y is a set of real-valued vectors $\{v_1, \ldots, v_n\}$ such that $Y_{ij} = v_i^T v_j$ for all i, j (i.e., $Y = B^T B$ for a corresponding matrix B). It is well known that a matrix Y is positive semidefinite if and only if it has a Gram matrix representation.

2. The Lovász–Schrijver matrix-cut operators. In this section we describe the so-called matrix-cut operators that were proposed by Lovász and Schrijver [23]. Given a convex set (e.g., a polytope) P, the matrix-cut operators consider P as a relaxation of the convex hull of its 0-1 vectors and produce another relaxation that is tighter than P. In other words, these operators produce a convex set that is sandwiched (in terms of containment) between P and (the convex hull of) the 0-1 vectors in P. Furthermore, the produced relaxation is strictly tighter than P, unless P is already tight. Our description and notation mostly follow that of Lovász and Schrijver [23] (but also those of [5, 10]). An alternative formulation of the matrix-cut operators is given by Lovász in [22].

Section 2.1 reviews the definitions of the Lovász–Schrijver matrix-cut operators. In section 2.2 we state some of their known properties (that we need), focusing on the application of these operators to the stable set problem. For completeness (and to aid readers who are unfamiliar with these operators), we give the proofs of these properties in the appendix, where these and relevant known results and examples are repeated and extended to a more general setting that includes the $N_{\mathrm{FR}+}$ operator.

Throughout, let e_j be the jth unit vector, let $\mathbf{0}$ be the vector of all zeros, and let $\mathbf{1} = \sum_j e_j$ be the vector of all ones. The sizes (dimensions) of $\mathbf{0}, \mathbf{1}$, and e_j will be clear from the context. Recall that a set is called a *cone* if it is closed under multiplication by a nonnegative number. A *convex cone* is thus a set that is closed under a nonnegative linear (i.e., *conic*) combination. (Throughout, we will consider convex cones rather than polytopes.) A *polyhedral cone* is a cone that is also a polyhedron; equivalently, a polyhedral cone is a set that can be defined by $\{x : Ax \geq 0\}$ for some matrix A.

2.1. Definitions.

Homogenization. It will be convenient to deal with homogenous systems of inequalities. We therefore embed the n-dimensional space $\mathbb{R}^n$ in $\mathbb{R}^{n+1}$ as the hyperplane $x_0 = 1$ (throughout, the 0th variable plays a special role) and work with convex cones in $\mathbb{R}^{n+1}$, as follows.

Since we deal with 0-1 programming on n variables, our basic example is a polytope P that is contained in $[0,1]^n$ (the convex hull of the n-dimensional hypercube $\{0,1\}^n$). To homogenize P using the new variable x_0, first embed P in the hyperplane $x_0 = 1$ of $\mathbb{R}^{n+1}$ and then generate from it a convex cone. That is, if

$$P = \{x \in \mathbb{R}^n : Ax \le b,\ \mathbf{0} \le x \le \mathbf{1}\}, \tag{1}$$

then the convex cone obtained by homogenization is

$$K := \left\{ \begin{pmatrix} x_0 \\ x \end{pmatrix} \in \mathbb{R}^{n+1} : Ax \le x_0 b,\ 0 \le x \le x_0 \mathbf{1} \right\}. \tag{2}$$

Note that such K can be described as the intersection of finitely many halfspaces defined by linear constraints $u^t x \ge 0$ (here $x \in \mathbb{R}^{n+1}$), and hence it is a polyhedral cone.

We denote by $Q \subset \mathbb{R}^{n+1}$ the convex cone that is obtained from the polytope $[0,1]^n$ via the homogenization procedure (1)–(2). Namely,

$$Q := \left\{ (x_0, x_1, \ldots, x_n)^T : 0 \le x_i \le x_0 \text{ for all } 1 \le i \le n \right\}. \tag{3}$$

Note that Q is a polyhedral cone that can be described by $2n$ linear inequalities.

Throughout, let $K \subseteq Q$ be a (closed) convex cone. We denote by K_I the convex cone that is generated by all 0-1 vectors in K. Observe that within the hyperplane $x_0 = 1$, K_I is exactly the *integral hull* (i.e., convex hull of the integral vectors) of K. For example, $Q_I = Q$.

The *polar cone* of K, denoted K^*, is the convex cone defined by

$$K^* := \{u \in \mathbb{R}^{n+1} : x^T u \ge 0 \text{ for all } x \in K\}.$$

Observe that a vector $u \in K^*$ corresponds to a linear constraint $u^T x \ge 0$ that is *valid* for K (i.e., satisfied by all vectors $x \in K$). The polar cone K^* is thus the collection of valid linear constraints for K. For example, Q is defined in (3) by $2n$ linear constraints, and hence Q^* is spanned by the vectors e_i and $f_i = e_0 - e_i$ for $i = 1, \ldots, n$.

Fractional stable sets. We will be mostly interested in the stable set problem. Let $G(V,E)$ be a graph with no isolated vertices and $|V| = n$. Then the stable sets of G correspond to the 0-1 solutions of the system of linear inequalities

$$x_i \ge 0 \quad \text{for all } i \in V \qquad (\textit{nonnegativity constraints}) \tag{4}$$

and

$$x_i + x_j \le 1 \quad \text{for all } ij \in E \qquad (\textit{edge constraints}). \tag{5}$$

Let $\mathrm{STAB}(G) \subset \mathbb{R}^n$ denote the convex hull of the 0-1 solutions of the system (4)–(5). Let $\mathrm{FRAC}(G) \subset \mathbb{R}^n$ (for "fractional stable sets") denote the solution set of the system (4)–(5) (i.e., without integrality restriction). Clearly, $\mathrm{STAB}(G) \subseteq \mathrm{FRAC}(G)$.

Let $\mathrm{FR}(G) \subset \mathbb{R}^{n+1}$ be the polyhedral cone that is obtained from the polytope $\mathrm{FRAC}(G)$ via the homogenization procedure (1)–(2). That is, $\mathrm{FR}(G)$ is the solution set of the following homogenous system of linear inequalities for the stable set problem:

$$x_i \geq 0 \quad \text{for each } i \in V, \tag{6}$$

$$x_0 - x_i - x_j \geq 0 \quad \text{for each } ij \in E. \tag{7}$$

Let $\mathrm{ST}(G)$ be the polyhedral cone that is obtained from the polytope $\mathrm{STAB}(G)$ via the homogenization procedure (1)–(2). It is straightforward that $(\mathrm{FR}(G))_I = \mathrm{ST}(G)$.

Throughout, we omit the graph G when it is clear from the context, denoting $\mathrm{STAB}(G)$ by STAB, etc. It can be seen that the polar cone FR^* is spanned by the vectors e_i for $i = 1, \ldots, n$ and the vectors $f_{ij} = e_0 - e_i - e_j$ for $ij \in E$. Note that $\mathrm{FR} \subseteq Q$ and hence $\mathrm{FR}^* \supseteq Q^*$.

Matrix-cut operators. Let $K_1, K_2 \subseteq Q$ be closed convex cones in $\mathbb{R}^{n+1}$ (e.g., $K_1 = \mathrm{FR}(G)$ and $K_2 = Q$). Consider the cone $K_1 \cap K_2$. For each $u \in K_1^*$ the constraint $u^T x \geq 0$ is valid for K_1, and for each $v \in K_2^*$ the constraint $v^T x \geq 0$ is valid for K_2. It follows that the quadratic inequality $(u^T x)(x^T v) \geq 0$ is valid for $K_1 \cap K_2$. Furthermore,

$$K_1 \cap K_2 = \left\{x : u^T x x^T v \geq 0 \text{ for all } u \in K_1^*, v \in K_2^*, x_0 \geq 0\right\}$$

because any original inequality, say $u^T x \geq 0$ for K_1, can be recovered by adding the two quadratic inequalities obtained by $e_i, f_i \in Q^* \subseteq K_2^*$, giving $u^T x \cdot x_0 = u^T x x^T (e_i + f_i) \geq 0$.

Furthermore, all 0-1 vectors in $K_1 \cap K_2$ satisfy $x_i^2 = x_i$. Therefore, if x is a 0-1 vector in $K_1 \cap K_2$ and with $x_0 = 1$, then setting $Y = xx^T$ we have the following:

(a) Y is symmetric.
(b) $Ye_0 = \mathrm{diag}(Y)$, i.e., $Y_{ii} = Y_{i0}$ for all $1 \leq i \leq n$.
(c) $u^T Y v \geq 0$ for all $u \in K_1^*$ and $v \in K_2^*$.
(d) Y is positive semidefinite.

Note that (c) can be written as

(c′) $YK_2^* \subseteq K_1$.

Lovász and Schrijver [23] proposed the following lift-and-project procedure. Given K_1, K_2, consider the derived cones

$$M(K_1, K_2) := \{Y \in \mathbb{R}^{(n+1)\times(n+1)} : Y \text{ satisfies (a)–(c)}\},$$

$$M_+(K_1, K_2) := \{Y \in \mathbb{R}^{(n+1)\times(n+1)} : Y \text{ satisfies (a)–(d)}\}$$

and define the projections of these liftings on $\mathbb{R}^{n+1}$:

$$N(K_1, K_2) := \{Ye_0 : Y \in M(K_1, K_2)\},$$

$$N_+(K_1, K_2) := \{Ye_0 : Y \in M_+(K_1, K_2)\}.$$

It follows from the above discussion that

$$(K_1 \cap K_2)_I \subseteq N_+(K_1, K_2) \subseteq N(K_1, K_2) \subseteq K_1 \cap K_2. \tag{8}$$

Relevant variants of the operators. We shorten notation to better handle two important special cases. When $K_2 = Q$ we omit K_2, i.e., $N(K) := N(K, Q)$ and $N_+(K) := N_+(K, Q)$. In this case, we have that (c′) is equivalent to the following:

(c″) Every column of Y is in K_1; the difference of the first column and any other column of Y is in K_1.

Note that we have from (8) that

$$K_I \subseteq N_+(K) \subseteq N(K) \subseteq K. \tag{9}$$

For the stable set problem, we may take $K_2 = \mathrm{FR}$, denoting it in the subscript, i.e., $N_{\mathrm{FR}}(K) := N(K, FR)$ and $N_{\mathrm{FR}+}(K) := N_+(K, FR)$. In this case, we have that (c′) is equivalent to the following:

(c‴) $Ye_i \in K_1$ for all $i \geq 1$, and $Yf_{ij} \in K_1$ for all $ij \in E$.

We assume throughout that $K \subseteq \mathrm{FR}$, and then we have from (8) that

$$K_I \subseteq N_{\mathrm{FR}+}(K) \subseteq N_{\mathrm{FR}}(K) \subseteq K. \tag{10}$$

It follows from the definition that using $K_2 = \mathrm{FR}$ is at least as strong as using $K_2 = Q$ in the same operator, i.e., $N_{\mathrm{FR}}(K) \subseteq N(K)$ and $N_{\mathrm{FR}_+}(K) \subseteq N_+(K)$. We therefore have that

$$K_I \subseteq N_{\mathrm{FR}+}(K) \subseteq N_{\mathrm{FR}}(K) \subseteq N(K) \subseteq K, \tag{11}$$

$$K_I \subseteq N_{\mathrm{FR}+}(K) \subseteq N_+(K) \subseteq N(K) \subseteq K. \tag{12}$$

It can also be seen that $N_{\mathrm{FR}}(K) \not\subseteq N_+(K)$ (e.g., when G is a clique on 5 vertices and taking $K = \mathrm{FR}$; see Appendix A.2), but it is not clear (to us) whether $N_+(K) \subseteq N_{\mathrm{FR}}(K)$. The strength of these operators is further discussed in Appendix A.2.

Iterated operators. Define the iterated operator $N^r(K)$ recursively by $N^0(K) = K$ and $N^r(K) = N(N^{r-1}(K))$ for $r \geq 1$. For other operators, the iterated operator is defined similarly.

The following theorem of Lovász and Schrijver [23] proves that even without the positive semidefiniteness constraint (d), it suffices to apply n iterations in order to get from a convex cone $K \subseteq Q$ the cone K_I. It follows that applying the N operator on $K \neq K_I$ produces a relaxation of K_I that is strictly tighter than K.

THEOREM 2.1 (Lovász and Schrijver [23]). *Let $K \subseteq Q$ be a convex cone in $\mathbb{R}^{n+1}$. Then $N^n(K) = K_I$.*

It is often easier to work in the original n-dimensional space (without homogenization), so in the case that K is the cone obtained from a polytope (or a convex set) P in $[0,1]^n$ via the homogenization procedure (1)–(2), define

$$N(P) := \left\{ x \in \mathbb{R}^n : \begin{pmatrix} 1 \\ x \end{pmatrix} \in N(K) \right\},$$

and similarly for the other operators (including the iterated ones).

For the stable set problem, K will be one of the cones obtained from $\mathrm{FR}(G)$ by an iterated operator, e.g., $N^r(\mathrm{FR}(G))$. Going back to the original n-dimensional space we shall abbreviate $N^r(G) := N^r(\mathrm{FRAC}(G))$, and similarly for the other operators. We then have from Theorem 2.1 that $N^n(G) = \mathrm{STAB}(G)$.

Ranks. The *N-rank of an inequality* $u^T x \geq 0$ that is valid for K_I is the smallest nonnegative integer r such that $u^T x \geq 0$ is valid for $N^r(K)$. (Note that the rank is relative to K.) For N_+, N_{FR}, and $N_{\mathrm{FR}+}$ the rank is defined similarly. Theorem 2.1 implies that the rank of any valid inequality is at most n (the dimension).

The *N-rank of a cone* K is the smallest nonnegative integer r such that $N^r(K) = K_I$, and similarly for the other operators. By Theorem 2.1, the N-rank of K is at most n (the dimension).

The N-rank of a graph G is the N-rank of $\mathrm{FR}(G)$, and similarly for the other operators. For example, for a bipartite graph, STAB = FRAC, and hence the N-rank of a bipartite graph is 0. We discuss bounds on the rank in Appendix A.2.

2.2. Useful properties.

Algorithmic aspects. Lovász and Schrijver [23] give sufficient conditions for efficient *weak* (i.e., up to arbitrary precision) optimization (of linear objective functions) over $N(K)$, $N_+(K)$, $N_{\mathrm{FR}}(K)$, and $N_{\mathrm{FR}+}(K)$. Technically, the matrix-cut operators have the following algorithmic property.

THEOREM 2.2 (Lovász and Schrijver [23]). *A polynomial time weak separation oracle for K gives a polynomial time weak separation oracle for $N^r(K), N_+^r(K), N_{\mathrm{FR}}^r(K)$, and $N_{\mathrm{FR}+}^r(K)$ for any fixed constant r.*

By the equivalence between weak (i.e., up to arbitrary precision) optimization and weak separation (see [11]), Theorem 2.2 implies a weak optimization of any linear objective function over these relaxations of K_I.

Lovász and Schrijver [23] suspect that Theorem 2.2 does not extend to $N(K, K)$. They remark, however, that if K is given by an explicit system of polynomially many linear inequalities, then Theorem 2.2 does extend to $N(K, K)$.

For the stable set problem, the cone $K = \mathrm{FR}$ is given by an explicit linear program of polynomial size, so one can solve the separation problem for it in polynomial time. We thus obtain the following theorem.

THEOREM 2.3. *For every fixed $r \geq 0$, the weak optimization problem for $N^r(G)$ can be solved in polynomial time, and similarly for $N_+^r, N_{\mathrm{FR}}^r, N_{\mathrm{FR}+}^r$.*

Down-monotonicity. A nonempty convex set $P \subseteq [0,1]^n$ is called *down-monotone* (in $[0,1]^n$) if for every $x \in P$, every $y \in [0,1]^n$ with $y \leq x$ is also in P (see, e.g., [11, p. 11]). Similarly, a convex cone $\{\mathbf{0}\} \neq K \subseteq Q$ is called *down-monotone* if for every $x \in K$, every $y \in Q$ with $y \leq x$ and $y_0 = x_0$ is also in K.

The next lemma shows that the relaxations of the stable set problem that are produced by iterated matrix-cut operators are down-monotone. Its proof appears in Appendix A.1.

LEMMA 2.4. *$N^r(G)$ is down-monotone for every $r \geq 0$, and similarly for N_+^r, N_{FR}^r, $N_{\mathrm{FR}+}^r$.*

Removing vertices from the graph. Recall that $V = \{1, \ldots, n\}$. For a vector $x \in \mathbb{R}^n$ and a subset $W \subset V$, we denote by x_W the restriction of x to the coordinates of W.

The next lemma characterizes the relaxations of the stable set problem that are produced by iterated matrix-cut operators when one of the coordinates is fixed (i.e., $x_i = 0$ or $x_i = x_0$). Its proof appears in Appendix A.1.

LEMMA 2.5. *Let $x \in \mathbb{R}^n$ and assume that i satisfies $x_i = 1$ and $x_j = 0$ for all $j \in \Gamma(i)$. Then for all $r \geq 0$, $x \in N^r(G)$ if and only if $x_{V-\Gamma(i)-i} \in N^r(G - \Gamma(i) - i)$, and similarly for N_+^r, N_{FR}^r, and $N_{\mathrm{FR}+}^r$.*

Vertex deletion and contraction. Let $a^T x \le b$ be an inequality valid for STAB(G). For a subset $W \subset V$, we denote by a_W the restriction of a to the coordinates of W. For every $i \in V$, if $a^T x \le b$ is valid for STAB(G), then $a^T_{V-i} x \le b$ is valid for STAB($G-i$) and $a^T_{V-\Gamma(v)-i} x \le b - a_i$ is valid for STAB($G - \Gamma(i) - i$). Following the terminology of Lovász and Schrijver [23], we say that these inequalities arise from $a^T x \le b$ by the *deletion* and *contraction* of vertex i, respectively. Note that if $a^T x \le b$ is an inequality such that for some i, both the deletion and the contraction of i yield inequalities valid for the corresponding graphs, then $a^T x \le b$ is valid for G.

Upper bounds on the N_+-rank. Lovász and Schrijver [23] prove the following bounds for the N_+ operator.

LEMMA 2.6 (Lovász and Schrijver [23]). *If $a^T x \le b$ is an inequality valid for* STAB(G) *such that for all $i \in V$ with $a_i > 0$ the contraction of i gives an inequality with N_+-rank at most r, then $a^T x \le b$ has N_+-rank at most $r+1$.*

LEMMA 2.7 (Lovász and Schrijver [23]). *The N_+-rank of a graph G is at most its stability number $\alpha(G)$.*

3. The Lovász–Schrijver relaxations in a random graph. In this section we show that the N_+-rank of a random graph $G_{n,1/2}$ is almost surely $\Theta(\log n)$. In particular, we analyze the asymptotic behavior of $\max\{\mathbf{1}^T x : x \in N_+^r(G)\}$ for $r = o(\log n)$. Loosely speaking, we show that the value of this relaxation is almost surely roughly $\sqrt{n/2^r}$. The precise formulations of our lower bound and upper bound on $\max\{\mathbf{1}^T x : x \in N_+^r(G)\}$ appear below. Our analysis extends the proof of Juhász [15] that shows that the theta function of a random graph is almost surely $\Theta(\sqrt{n})$.

THEOREM 3.1. *For any $c > \sqrt{2}$ there exists an $\epsilon' > 0$ such that if $0 \le r \le \epsilon' \log n$, then almost surely* $\max\{\mathbf{1}^T x : x \in N_+^r(G_{n,1/2})\} \ge \sqrt{n}/c^{r+1}$, *and similarly for $N^r_{\mathrm{FR}+}$.*

The proof of Theorem 3.1 appears in section 3.1. Technically, we show that $N_+^r(G_{n,1/2})$ almost surely contains the "uniform" solution $(1/c^{r+1}\sqrt{n})\mathbf{1}$ and hence obtain a lower bound on the probable value of the relaxation.

To show that the above lower bound is nearly tight, we next give an upper bound on the value of the relaxation. Its proof appears in section 3.2.

THEOREM 3.2. *For any $d < \sqrt{2}$ there exists an $\epsilon' > 0$ such that if $1 \le r \le \epsilon' \log n$, then almost surely* $\max\{\mathbf{1}^T x : x \in N_+^r(G_{n,1/2})\} \le 4\sqrt{n}/d^{r+1}$, *and similarly for $N^r_{\mathrm{FR}+}$.*

It is straightforward that Theorem 1.1 follows from Theorems 3.1 and 3.2 by taking $c = \sqrt{2+\delta}$ and $d = \sqrt{2-\delta}$.

The N_+-rank of a random graph $G_{n,1/2}$. Using Theorem 3.1 and Lemma 2.7 we can now show that the N_+-rank of a random graph is almost surely $\Theta(\log n)$, proving Theorem 1.2. For comparison, it follows from Corollary A.24 that the N-rank of a random graph is almost surely at least $\Omega(n/\log n)$.

Proof of Theorem 1.2. Let G be a random graph from the distribution $G_{n,1/2}$, and let us first show a lower bound on the N_+-rank. It is well known that, almost surely, the maximum size of a stable set in G is roughly $2\log_2 n$, i.e.,

$$\max\{\mathbf{1}^T x : x \in \mathrm{STAB}\} \le O(\log n).$$

We have from Theorem 3.1 with $r = \epsilon' \log n$ that, almost surely,

$$\max\{\mathbf{1}^T x : x \in N_+^r(\mathrm{FRAC})\} \ge n^{\Omega(1)}.$$

It follows that $N_+^r(\mathrm{FRAC}) \ne \mathrm{STAB}$, and hence the N_+-rank of FRAC (and therefore of G) is larger than $r = \epsilon' \log n = \Omega(\log n)$.

The upper bound on the N_+-rank of G follows from Lemma 2.7. Indeed, the stability number of a random graph $G_{n,1/2}$ is almost surely roughly $2\log_2 n$, and hence the N_+-rank of G is almost surely $O(\log n)$, as claimed. □

3.1. Lower bound on the value of $N_+^r(G_{n,1/2})$. We prove Theorem 3.1 by showing that $N_+^r(G_{n,1/2})$ almost surely contains the "uniform" solution $(1/c^{r+1}\sqrt{n})\mathbf{1}$. First we exhibit in Lemma 3.3 certain conditions that are sufficient for such a uniform solution to be feasible in $N_+^r(G)$. We then show in Lemma 3.4 that these conditions are almost surely satisfied by a random graph $G_{n,1/2}$.

We will say that two vertices are *nonadjacent* if they are not adjacent and they are not equal (i.e., they are adjacent in the complement graph). We make no attempt to optimize constants.

LEMMA 3.3. *Let G be a graph on n vertices, let $c = \sqrt{2}(1+\epsilon)^{10}$ for $0 < \epsilon < 1/5$, and let $r \geq 0$. Assume that for every $S \subset V$ with $|S| \leq r$, the graph $G' = G - S - \Gamma(S)$ satisfies the following (letting n' denote the number of vertices in G'):*

(i) *All eigenvalues of the adjacency matrix of $\overline{G'}$ are at least $-(1+\epsilon)\sqrt{n'}$.*

(ii) *The degree of every vertex in $\overline{G'}$ is between $\frac{1}{1+\epsilon}\frac{n'}{2}$ and $(1+\epsilon)\frac{n'}{2}$.*

If $c^{r+1} \leq \epsilon\sqrt{n}$, then $(1/c^{r+1}\sqrt{n})\mathbf{1} \in N_+^r(G)$, and similarly for $N_{\mathrm{FR}+}^r(G)$.

Proof. Proceed by induction on r. For the base case $r = 0$, observe that $(1/c^{r+1}\sqrt{n})\mathbf{1}$ (and even $(1/2)\mathbf{1}$) satisfies the nonnegativity and edge constraints and therefore is in $\mathrm{FR}(G)$ by definition.

For the inductive step, assume that it holds for $r \geq 0$, and let us show that it holds for $r+1$. Let G be a graph with (i) and (ii) holding for any $|S| \leq r+1$, and $c^{r+2} \leq \epsilon\sqrt{n}$. We can choose, in particular, $|S| = 0$ and have that (i) and (ii) hold for the graph G itself. To ease notation, define

$$\mu := (1+\epsilon)^5(c^{r+1}/\sqrt{2})\sqrt{n}. \tag{13}$$

Let A be the $n \times n$ adjacency matrix of $\overline{G}$, i.e., $A_{ij} = 0$ whenever $(i,j) \in E$ or $i = j$, and $A_{ij} = 1$ otherwise. We know from (i) that all eigenvalues of A are at least $-(1+\epsilon)\sqrt{n} \geq -\mu$. Hence, the matrix $B = A + \mu I$ is positive semidefinite, and there exist vectors $z_1, \ldots, z_n$ such that $B_{ij} = z_i^T z_j$. Therefore

$$\|z_i\|^2 = B_{ii} = \mu \quad \text{for all } i \geq 1. \tag{14}$$

Let $z_0 = \sum_{i=1}^n z_i$. Then

$$\|z_0\|^2 = \left(\sum_{i>0} z_i\right)^T \left(\sum_{j>0} z_j\right) = \sum_{i,j>0} B_{ij} = \sum_{i>0}\sum_{j>0} B_{ij}.$$

To estimate $\sum_{j>0} B_{ij} = \sum_{j>0} A_{ij} + \mu$ for $i > 0$, observe that we have from (ii) that

$$\frac{1}{1+\epsilon}\frac{n}{2} \leq \sum_{j>0} A_{ij} \leq (1+\epsilon)\frac{n}{2},$$

while $\mu \leq (c^{r+2}/2)\sqrt{n} \leq \epsilon n/2$. Hence,

$$\frac{1}{1+\epsilon}\frac{n}{2} \leq \sum_{j>0} B_{ij} \leq (1+\epsilon)^2\frac{n}{2}, \tag{15}$$

and we conclude that

$$\frac{1}{1+\epsilon}\frac{n^2}{2} \le \|z_0\|^2 \le (1+\epsilon)^2\frac{n^2}{2}. \tag{16}$$

For every $i \ge 0$ let v_i be the unit length vectors in the direction of the vector z_i, i.e., $v_i = z_i/\|z_i\|$, and let $x_i = (v_i^T v_0)^2$. Observe that $x_0 = (v_0^T v_0)^2 = 1$.

We claim that $x = (x_1, \ldots, x_n)^T$ is in $N_+^{r+1}(G)$. Let us first show how the proof of Lemma 3.3 follows from this claim. Indeed, from (ii) we have that

$$v_i^T v_0 = \left(\frac{z_i}{\|z_i\|}\right)^T \left(\frac{\sum_{j>0} z_j}{\|z_0\|}\right) = \frac{\sum_{j>0} B_{ij}}{\sqrt{\mu}\|z_0\|}.$$

Together with (15) and (16) we can estimate $x_i = (v_i^T v_0)^2$ by

$$\frac{1}{(1+\epsilon)^4}\cdot\frac{1}{2\mu} \le x_i \le (1+\epsilon)^5\frac{1}{2\mu}, \tag{17}$$

and from (13) we have that

$$x_i \ge \frac{1}{2(1+\epsilon)^4}\cdot\frac{\sqrt{2}}{(1+\epsilon)^5 c^{r+1}\sqrt{n}} \ge \frac{1}{c^{r+2}\sqrt{n}};$$

thus $(1/c^{r+2}\sqrt{n})\mathbf{1} \le x \in N_+^{r+1}(G)$. By the monotonicity guaranteed in Lemma 2.4 we have $(1/c^{r+2}\sqrt{n})\mathbf{1} \in N_+^{r+1}(G)$, which indeed proves the inductive step.

We now prove the claim $x \in N_+^{r+1}(G)$ by presenting a matrix $Y \in M_+(N_+^r(G))$ whose 0th column corresponds to x. Indeed, let Y be the $(n+1)\times(n+1)$ matrix defined by $Y_{ij} = (v_i^T v_j)\sqrt{x_i x_j}$ for all $i, j \ge 0$. By definition, $Y_{i0} = (v_i^T v_0)\sqrt{x_i} = x_i$ for $i \ge 0$, and in particular $Y_{00} = x_0 = 1$. We will show that Y satisfies (a), (b), (c$''$), and (d). Three of them are straightforward:

(a) Y is symmetric by definition.
(b) $Y_{ii} = \|v_i\|^2 x_i = x_i$ and hence $Y_{ii} = x_i = Y_{i0}$.
(d) Y is positive semidefinite because it can be represented by the vectors $\{\sqrt{x_i}v_i\}$, i.e., $Y_{ij} = (\sqrt{x_i}v_i)^T(\sqrt{x_j}v_j)$ for all $i, j \ge 0$.

Before proving (c$''$), observe that for $i, j > 0$ we have

$$Y_{ij} = \left(\frac{z_i}{\|z_i\|}\right)^T \left(\frac{z_j}{\|z_j\|}\right)\sqrt{x_i x_j} = (1/\mu)B_{ij}\sqrt{x_i x_j},$$

and B_{ij} is either μ, 0, or 1. So for $i, j > 0$ we have

$$Y_{ij} = \begin{cases} x_i & \text{if } i = j, \\ 0 & \text{if } i \ne j \text{ and } ij \in E, \\ (1/\mu)\sqrt{x_i x_j} & \text{if } i \ne j \text{ and } ij \notin E, \end{cases}$$

and the estimate of (17) gives that $x_i \sim 1/2\mu$ and $\sqrt{x_i x_j} \sim 1/2\mu$. Hence,

$$Y = \left[\begin{array}{cccc} 1 & x_1 & \cdots & x_n \\ x_1 & x_1 & 0 & \frac{\sqrt{x_i x_j}}{\mu} \\ \vdots & & \ddots & \\ x_n & 0 & \frac{\sqrt{x_i x_j}}{\mu} & x_n \end{array}\right] \sim \left[\begin{array}{cccc} 1 & \frac{1}{2\mu} & \cdots & \frac{1}{2\mu} \\ \frac{1}{2\mu} & \frac{1}{2\mu} & 0 & \frac{1}{2\mu^2} \\ \vdots & & \ddots & \\ \frac{1}{2\mu} & 0 & \frac{1}{2\mu^2} & \frac{1}{2\mu} \end{array}\right].$$

Consider Ye_i, the ith column of Y, for $i > 0$, and scale it by a factor of $1/x_i$ so that its 0th entry will be 1. We get a fractional solution where vertex i has value 1, its adjacent vertices have value 0, and its nonadjacent vertices j have value $(1/\mu)\sqrt{x_j/x_i} \sim 1/\mu$. Let G' be the subgraph of G induced on the latter vertices (i.e., those nonadjacent to i), and let n' denote the number of vertices in G'. Then by Lemma 2.5, we have that the fractional solution Ye_i is in $N_+^r(G)$ if and only if its restriction to G' is in $N_+^r(G')$. Each coordinate in the fractional solution restricted to G' is bounded by

$$\frac{1}{\mu}\sqrt{\frac{x_j}{x_i}} \leq \frac{1}{\mu}(1+\epsilon)^{9/2} \leq \frac{\sqrt{2}}{c^{r+1}\sqrt{n(1+\epsilon)}} \leq \frac{1}{c^{r+1}\sqrt{n'}},$$

where the first inequality is due to (17), the second is due to (13), and the third follows from $n' \leq (1+\epsilon)\frac{n}{2}$, which we have from (ii). The fractional solution restricted to G' is thus dominated by the uniform solution $(1/c^{r+1}\sqrt{n'})\mathbf{1}$, which belongs to $N_+^r(G')$ by applying the induction hypothesis to G'. (Note that G' satisfies (i) and (ii) for any $0 \leq |S| \leq r$ by definition, and that we have $c^{r+1} \leq \epsilon\sqrt{n}/c \leq \epsilon\sqrt{n'}$.) From the monotonicity guaranteed by Lemma 2.4, we conclude that also the fractional solution restricted to G' is in $N_+^r(G')$, and therefore $Ye_i \in N_+^r(G)$.

Consider Yf_i, the difference between column 0 and column i of Y, for $i > 0$. Its 0th entry is $1 - x_i \sim 1 - 1/2\mu$, its ith entry is 0, and any other jth entry is at most roughly $1/2\mu$. Observe that

$$x_i \leq \frac{(1+\epsilon)^5}{2\mu} \leq \frac{1}{\sqrt{2n}} \leq 1 - \frac{1}{\sqrt{2}}, \tag{18}$$

where the first inequality is due to (17), the second is due to (13), and the third is due to $\sqrt{n} \geq 5\epsilon\sqrt{n} \geq 5c^{r+2} > 10$. Scaling the vector Yf_i by a factor $1/(1-x_i)$ so that its 0th entry is 1, we obtain a fractional solution in which the value of the jth entry is at most

$$\frac{x_j}{1-x_i} \leq \frac{(1+\epsilon)^5/2\mu}{1/\sqrt{2}} = \frac{1}{c^{r+1}\sqrt{n}}.$$

The fractional solution is thus dominated by $(1/c^{r+1}\sqrt{n})\mathbf{1}$, which by the induction hypothesis belongs to $N_+^r(G)$. (Note that G satisfies the requirements for r). From the monotonicity guaranteed by Lemma 2.4 (as all entries of Yf_i are nonnegative), we conclude that $Yf_i \in N_+^r(G)$.

We therefore have that (c″) holds, which completes the proof of the inductive step for $N_+^r(G)$.

Finally, let us show that the proof extends also to $N_{\mathrm{FR}+}^r(G)$. We need to consider also Yf_{ij} for $ij \in E$. The 0th entry of this vector is $1 - x_i - x_j \sim 1 - 2/2\mu$, the ith and jth entries are 0, and any other kth entry is either roughly $1/2\mu$ if k is adjacent to both i, j, or roughly $1/2\mu - 2/2\mu^2 \sim 1/2\mu$ if k is nonadjacent to both i, j, or roughly $1/2\mu - 1/2\mu^2 \sim 1/2\mu$ if k is adjacent to exactly one of i, j. Similar to (18) we have that

$$x_i + x_j \leq 2 \cdot \frac{1}{\sqrt{2n}} \leq 1 - \frac{1}{\sqrt{2}}.$$

Scaling this vector (by a small factor) so that the 0th entry is 1, we obtain a fractional

solution in which the value of the kth entry is at most

$$\frac{x_k}{1-x_i-x_j} \le \frac{(1+\epsilon)^5/2\mu}{1/\sqrt{2}} = \frac{1}{c^{r+1}\sqrt{n}}.$$

The fractional solution is thus dominated by $(1/c^{r+1}\sqrt{n})\mathbf{1}$, which by the induction hypothesis belongs to $N_+^r(G)$. From the monotonicity guaranteed by Lemma 2.4 (as all entries of Yf_{ij} are nonnegative), we conclude that $Yf_{ij} \in N_+^r(G)$. □

LEMMA 3.4. *Let $\epsilon > 0$. Then there exists an $\epsilon' > 0$ that depends only on ϵ such that for any $r \le \epsilon' \log n$, a random graph $G_{n,1/2}$ almost surely satisfies all the requirements of Lemma* 3.3.

Proof. Observe that a sufficiently small $\epsilon' > 0$ that depends on ϵ guarantees that $c^{r+1} \le \epsilon\sqrt{n}$ (we can assume, without loss of generality, that $\epsilon < 1/5$).

Consider a particular choice of S of size $s \le r$, and its corresponding graph $G'(V', E')$ (the subgraph of G induced on the vertices that are nonadjacent to all the vertices of S). The number of vertices in G', which we denote by $n' = |V'|$, has binomial distribution $B(n-s, 1/2^s)$. Since $s \le \log n \le n/4$, we have by Chernoff bound that

$$\mathbb{P}\left[n' \le n/2^{s+1}\right] \le 2^{-\delta_1 n/2^s} \tag{19}$$

for some fixed $\delta_1 > 0$.

G' is a random graph (with edge probability $1/2$) on n' vertices. Therefore, the adjacency matrix of $\overline{G'}$ is a random symmetric matrix, and we can use results on the concentration of its eigenvalues. In particular, we have from Krivelevich and Vu [19] (who improve the concentration shown by Füredi and Komlós [9]; see also [2]) that

$$\mathbb{P}\left[G' \text{ does not satisfy (i)}\right] \le 2^{-\delta_2 n'} \tag{20}$$

for some $\delta_2 > 0$ that depends on ϵ.

Since G' is a random graph, the degree of a particular vertex in $\overline{G'}$ has binomial distribution $B(n'-1, 1/2)$. By Chernoff bound and the union bound on the n' vertices we have that

$$\mathbb{P}\left[G' \text{ does not satisfy (ii)}\right] \le n' 2^{-\delta_3 n'} \tag{21}$$

for some fixed $\delta_3 > 0$ that depends on ϵ.

Using the union bound on the events of (20) and (21) we can bound the probability that G' does not satisfy (i) or (ii). In order to obtain a bound in terms of n (rather than n'), we add to the union bound also the event of (19) and have that for some fixed $\delta > 0$ that depends on ϵ,

$$\mathbb{P}\left[G' \text{ does not satisfy (i) or (ii)}\right] \le n 2^{-\delta n/2^s}.$$

Taking the union bound on all possible sets S of size at most r, the probability that the requirements of Lemma 3.3 do not hold is at most

$$\sum_{s=0}^{r} \binom{n}{s} n 2^{-\delta n/2^s} \le r n^{r+1} 2^{-\delta n/2^r} \le n^{r+2} 2^{-\delta n/2^r} \ll 1$$

when $r \le \epsilon' \log n$ for a sufficiently small $\epsilon' > 0$ that depends on ϵ, and hence these requirements hold almost surely. □

The proof of Theorem 3.1 follows from Lemmas 3.3 and 3.4.

3.2. Upper bound on the value of $N_+^r(G_{n,1/2})$. We prove Theorem 3.2 by showing that the inequality $\mathbf{1}^T x \le 4\sqrt{n}/d^{r+1}$ is almost surely valid for $N_+^r(G)$. First we exhibit in Lemma 3.5 certain conditions that are sufficient for this inequality to be valid for $N_+^r(G)$. We then show in Lemma 3.6 that these conditions are almost surely satisfied by a random graph $G_{n,1/2}$.

The Lovász theta function of a graph is defined as $\vartheta(G) := \max\{\mathbf{1}^T x : x \in TH(G)\}$, where $TH(G)$ is the solution set of the nonnegativity constraints (4) and the so-called orthogonality constraints (see [21, 11] for a definition). Lovász and Schrijver [23] show that the orthogonality constraints have N_+-rank at most 1, and hence $N_+(G) \subseteq TH(G)$.

LEMMA 3.5. *Let G be a graph on n vertices, let $d = \sqrt{2}(1-\epsilon)$ for $0 < \epsilon < 1$, and let $r \ge 1$. Assume that for every $S \subset V$ with $|S| \le r$, the graph $G' = G - S - \Gamma(S)$ satisfies the following (letting n' denote the number of vertices in G'):*

(i) $\vartheta(G') \le 2(1+\epsilon)\sqrt{n'}$.

(ii) *The degree of every vertex in $\overline{G'}$ is between $\frac{1}{1+\epsilon}\frac{n'}{2}$ and $(1+\epsilon)\frac{n'}{2}$.*

If $d^{r+1} \le \epsilon^2\sqrt{n}$, then $\max\{\mathbf{1}^T x : x \in N_+^r(G)\} \le 4\sqrt{n}/d^{r+1}$, and similarly for $N_{\mathrm{FR}+}^r$.

Proof. Proceed by induction on r. For the base case $r = 1$, we can choose $|S| = 0$ and then (i) and (ii) hold for the graph G itself. In particular, we have that

$$\max\{\mathbf{1}^T x : x \in N_+(G)\} \le \vartheta(G) \le 2(1+\epsilon)\sqrt{n} < 4\sqrt{n}/d^2.$$

For the inductive step, assume that it holds for $r \ge 1$, and let us show that it holds for $r+1$. In other words, given a graph G with (i) and (ii) holding for any $|S| \le r+1$, we will prove that the inequality $\mathbf{1}^T x \le 4\sqrt{n}/d^{r+2}$ is valid for $N_+^{r+1}(G)$. By Lemma 2.6 we know that it suffices to prove that for every vertex v, the inequality that arises from the contraction of v, i.e., $\mathbf{1}^T x \le 4\sqrt{n}/d^{r+2} - 1$, is valid for $N_+^r(G - \Gamma(v) - v)$.

By the induction hypothesis for $G' = G - \Gamma(v) - v$ we have that $\max\{\mathbf{1}^T x : x \in N_+^r(G')\} \le 4\sqrt{n'}/d^{r+1}$, i.e., the inequality $\mathbf{1}^T x \le 4\sqrt{n'}/d^{r+1}$ is valid for $N_+^r(G')$. Since (ii) holds also for G itself, we have that $n' \le (1+\epsilon)\frac{n}{2}$, and hence

$$\frac{4\sqrt{n'}}{d^{r+1}} \le \frac{4\sqrt{n}}{d^{r+1}}\frac{\sqrt{1+\epsilon}}{\sqrt{2}} = \frac{4\sqrt{n}}{d^{r+2}}\sqrt{1+\epsilon}(1-\epsilon) \le \frac{4\sqrt{n}(1-\epsilon^2)}{d^{r+2}} \le \frac{4\sqrt{n}}{d^{r+2}} - 1,$$

where the last inequality follows from $d^{r+2} \le 4\epsilon^2\sqrt{n}$. Therefore we have that for $N_+^r(G')$ the inequality $\mathbf{1}^T x \le 4\sqrt{n'}/d^{r+1} \le 4\sqrt{n}/d^{r+2} - 1$ holds, which completes the proof of the inductive step.

Finally, the proof immediately extends to the $N_{\mathrm{FR}+}^r$ operator since $N_{\mathrm{FR}+}^r(G) \subseteq N_+^r(G)$. ∎

LEMMA 3.6. *Let $\epsilon > 0$. Then there exists an $\epsilon' > 0$ that depends only on ϵ such that for any $r \le \epsilon' \log n$, a random graph $G_{n,1/2}$ almost surely satisfies all the requirements of Lemma* 3.5.

Proof. The proof is similar to that of Lemma 3.4, but with the different requirement (i). Juhász [15] shows that $\vartheta(G')$ is at most $(2 + o(1))\sqrt{n'}$, almost surely, by using the result of Füredi and Komlós [9] on the concentration of eigenvalues of random symmetric matrices. By using the stronger concentration result of Krivelevich and Vu [19] (see also [2]), we have that the analogue of (20) holds, and the proof follows. ∎

The proof of Theorem 3.2 follows from Lemmas 3.5 and 3.6.

Appendix. Properties of the matrix-cut operators.

A.1. Basic properties. We collect some properties of the matrix-cut operators defined in section 2.1. In particular, we prove Lemmas 2.4 and 2.5 (which are used in section 3.1).

Monotonicity. It is straightforward that the matrix-cut operators are monotone with respect to containment of K_1 and K_2, as follows.

LEMMA A.1. *Let $K_1' \subseteq K_1$ and $K_2 \subseteq K_2'$. Then $N(K_1', K_2') \subseteq N(K_1, K_2)$, and similarly for N_+.*

For the stable set problem it follows that the matrix-cut operators are monotone with respect to adding/removing edges.

COROLLARY A.2. *Let G' be a graph that is obtained from another graph G by adding edges. Then $N^r(G') \subseteq N^r(G)$, and similarly for N_+^r, N_{FR}^r, $N_{\mathrm{FR}+}^r$.*

Proof. Observe that $\mathrm{FR}(G') \subseteq \mathrm{FR}(G)$. The proof follows from Lemma A.1. ∎

Down-monotonicity. The next lemma shows that down-monotonicity (see section 2.2 for a definition) is preserved by the matrix-cut operators. It extends a similar result that is given for $N(\cdot)$ and $N_+(\cdot)$ by Goemans and Tunçel [10, Theorem 5.1] (under the name lower-comprehensive) and by Cook and Dash [5, Lemma 2.6] (under the name anti-blocking type).

LEMMA A.3. *Let $K_1, K_2 \subseteq Q$ be down-monotone convex cones. Then $N(K_1, K_2)$ is down-monotone, and similarly for N_+.*

Proof. Let $x \in N(K_1, K_2)$ and $0 \le x' \le x$ with $x_0' = x_0$. It suffices to prove that $x' \in N(K_1, K_2)$ when x, x' differ only in a single coordinate, say $i = 1$, since we can repeat the same argument for each coordinate. Furthermore, for a single coordinate $i = 1$ it suffices to prove the case $x_1' = 0$, since $N(K_1, K_2)$ is convex, and so convex combinations of x' and x give any desired value in coordinate $i = 1$.

Since $x \in N(K_1, K_2)$, there exists a matrix $Y \in M(K_1, K_2)$ with $x = Ye_0$. Define the matrix Y' by

$$Y'_{ij} = \begin{cases} 0 & \text{if } i = 1 \text{ or } j = 1, \\ Y_{ij} & \text{otherwise.} \end{cases}$$

We claim that $Y' \in M(K_1, K_2)$. Indeed, Y' clearly satisfies (a) and (b). To prove (c), let $u \in K_1^*, v \in K_2^*$, and from Proposition A.4 below we have that $u - u_1 x_1 \in K_1^*$ and $v - v_1 x_1 \in K_2^*$, and hence

$$u^T Y' v = (u - u_1 x_1)^T Y (v - v_1 x_1) \ge 0.$$

Observe that $x' = Y' e_0$, and therefore $x' \in N(K_1, K_2)$, as required.

For the proof of N_+ we need to show that (d) also holds, and indeed from the Gram matrix representation of Y we can obtain a Gram matrix representation of Y' by replacing the vector that corresponds to coordinate $i = 1$ with the all-zeros vector $\mathbf{0}$. ∎

PROPOSITION A.4. *Let $K \subseteq Q$ be down-monotone and let $v \in K^*$. Then $v - v_i e_i \in K^*$ for all $i \ge 1$.*

Proof. By the down-monotonicity of K, for every $x \in K$ we have that $x - x_i e_i \in K$, and hence $(v - v_i e_i)^T x = \sum_{j \ne i} v_j x_j = v^T (x - x_i e_i) \ge 0$. ∎

We can now prove Lemma 2.4, i.e., show that $N^r(G)$ is down-monotone for every $r \ge 0$, and similarly for N_+^r, N_{FR}^r, $N_{\mathrm{FR}+}^r$.

Proof of Lemma 2.4. Observe that Q is down-monotone by its definition (3), and that FRAC is down-monotone by its definition (6)–(7). By Lemma A.3 the matrix-cut operators preserve down-monotonicity and the proof follows. ∎

Flipping and renaming coordinates. The operators $N, N_+, N_{\mathrm{FR}}, N_{\mathrm{FR}+}$ are invariant under various operations, including *renaming coordinates* (i.e., permuting the order of coordinates), and *flipping coordinates* $x_i \to (x_0 - x_i)$ for any subset of the coordinates $\{1, 2, \ldots, n\}$. More formally, we present the following lemma.

LEMMA A.5 (Lovász and Schrijver [23]). *Let A be a linear transformation mapping Q onto itself. Then $N(AK_1, AK_2) = AN(K_1, K_2)$, and similarly for N_+. Hence $N(AK) = AN(K)$, and similarly for N_+.*

By flipping coordinates, one can extend Lemma A.3. For example, it follows that the N and N_+ operators preserve up-monotonicity; see Cook and Dash [5, section 2] (as the blocking property) and Goemans and Tunçel [10, section 5] (as the "convex corner" property).

Intersection with faces. A *face* of Q is the intersection of Q with hyperplanes of the form $\{x : x_i = 0\}$ or $\{x : x_i = x_0\}$. The intersection of K with a face of Q consists of all $x \in K$ with one or more of their coordinates *fixed* to 0 or x_0 (recall that x_0 corresponds to 1 in the nonhomogenous case).

The following lemma proves equivalence between fixing some coordinates before applying a matrix-cut operator (e.g., in K) and afterwards (e.g., in $N(K)$). It extends a similar result that is given by Goemans and Tunçel [10] for $N(\cdot)$ and $N_+(\cdot)$.

LEMMA A.6. *If F is a face of Q, then $N(K_1 \cap F, K_2) = N(K_1, K_2) \cap F$ and similarly for N_+.*

Proof. The direction "$\subseteq$" follows from Lemma A.1, since $N(K_1 \cap F, K_2) \subseteq N(K_1, K_2)$ and $N(K_1 \cap F, K_2) \subseteq N(F, K_2) \subseteq F$, and similarly for N_+.

For the converse direction "$\supseteq$" with the N operator, let $x \in N(K_1, K_2) \cap F$. Then there exists a matrix $Y \in M(K_1, K_2)$ with $Ye_0 = x$. Let H be any one of the hyperplanes of the form $\{x : x_i = 0\}$ or $\{x : x_i = x_0\}$ that define F. Since $e_j, f_j \in Q^* \subseteq K_2^*$ for all j, we have that $Ye_j \in K_1 \subseteq Q$ and $Yf_j \in K_1 \subseteq Q$, while their sum satisfies $Ye_j + Yf_j = Ye_0 = x \in F \subset H$. Since H defines a face of Q, then by definition of a face we have that Ye_j (and also Yf_j) must belong to H.[1] But every $v \in \mathbb{R}^{n+1}$ is a linear combination of $\{e_0, e_1, \ldots, e_n\}$ and $Ye_j \in H$ for all $j \geq 0$, and so $Yv \in H$ for every v, including all $v \in K_2^*$.

For every $v \in K_2^*$ we have that Yv belongs to $K_1 \subseteq Q$ by the definition of Y. We saw above that Yv also belongs to all hyperplanes H that define F, and we conclude that Yv belongs also to F. Hence, $Yv \in K_1 \cap F$ for all $v \in K_2^*$, implying that $Y \in M(K_1 \cap F, K_2)$ and $x \in N(K_1 \cap F, K_2)$. The proof for N_+ is similar, since Y is also known to be positive semidefinite. □

We remark that the above proof of Lemma A.6 extends to the case where F is a face of K_1, as shown by Cook and Dash [5, Lemma 2.2] for $N(\cdot)$ and $N_+(\cdot)$. For the special cases $K_2 = Q$ and $K_2 = \mathrm{FR}$ we obtain the following.

COROLLARY A.7. *If F is a face of Q (or a face of K), then $N(K \cap F) = N(K) \cap F$, and similarly for N_+, N_{FR}, $N_{\mathrm{FR}+}$.*

Deleting fixed coordinates. Suppose that K is contained in a face of Q. Then some of the coordinates are fixed (i.e., $x_i = 0$ or $x_i = x_0$), and it may be desirable to delete these coordinates and reduce the dimension. Formally, a *deletion* operation of indices subset $I \subset \{1, \ldots, n\}$ is the function $f : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1-|I|}$, where $f(x)$ is the vector x restricted to the coordinates not in I, i.e., $f(x) = (x_i)_{i \notin I}$.

[1] In other words, suppose that the hyperplane H is defined by the equality $u^T x = 0$ (with $u = e_i$ or $u = f_i$) and that the inequality $u^T x \geq 0$ is valid for Q (i.e., Q is entirely contained in one side of H). We then have that $u^T(Ye_j), u^T(Yf_j) \geq 0$, while their sum is $u^T x = 0$, implying that $u^T(Ye_j) = u^T(Yf_j) = 0$.

For the stable set problem it is straightforward that the effect of fixing and deleting a coordinate of $\mathrm{FR}(G)$ is as follows.

LEMMA A.8. *Let $F = Q \cap \{x : x_i = 0\}$, and let f be the deletion operation of coordinate i. Then $f(\mathrm{FR}(G) \cap F) = \mathrm{FR}(G - i)$.*

LEMMA A.9. *Let $F = Q \cap \{x : x_i = x_0\}$, and let f be the deletion operation of coordinate i. Then $f(\mathrm{FR}(G) \cap F) = \mathrm{FR}(G - i) \cap \{x : \forall j \in \Gamma(i), x_j = 0\}$.*

We show below that deleting fixed coordinates of K before applying a matrix-cut operator (e.g., in K) is equivalent to deleting them afterwards (e.g., in $N(K)$). This extends similar results that are given for $N(\cdot)$ and $N_+(\cdot)$ by Cook and Dash [5] (see also [25]). Technically, they consider an embedding operation (that introduces new coordinates that are fixed to either 0 or x_0), which is just the inverse of the deletion operation.

We first handle the basic case of one coordinate that is fixed to 0 (Lemma A.10), then extend the result to an arbitrary face F and to an arbitrary K_2 (Lemma A.11), and finally specialize it to the cases $K_2 = Q$ and $K_2 = \mathrm{FR}$ (Corollary A.12).

LEMMA A.10. *Let $F = Q \cap \{x : x_n = 0\}$ and let f be the deletion operation of coordinate n. If $K_1, K_2 \subseteq F$ are convex cones, then $f(N(K_1, K_2)) = N(f(K_1), f(K_2))$,*[2] *and similarly for N_+.*

Proof. The deletion operation f is a linear transformation from $\mathbb{R}^{n+1}$ to $\mathbb{R}^n$ and thus can be described as an $n \times (n+1)$ matrix A. Note that columns 0 to $n-1$ of A form an identity matrix, and column n of A is all zeros. We first claim that $AK^* = (AK)^*$ for $K = K_1$ and for $K = K_2$. Indeed, by definition, $u \in AK^*$ if there exists $r \in \mathbb{R}$ with $\binom{u}{r} \in K^*$. Note that $\binom{u}{r} \in K^*$ holds either for all values of r or for no value of r, since $K \subset \{x : x_n = 0\}$. Therefore,

$$AK^* = \left\{ u : \exists r \in \mathbb{R} \text{ with } \begin{pmatrix} u \\ r \end{pmatrix} \in K^* \right\} = \left\{ u : \begin{pmatrix} u \\ 0 \end{pmatrix} \in K^* \right\}.$$

We also have that

$$(AK)^* = \{u : u^T(Ax) \geq 0 \ \ \forall x \in K\} = \{u : A^T u \in K^*\}.$$

Since $A^T u = \binom{u}{0}$, we obtain $AK^* = (AK)^*$.

Let us now prove that $M(AK_1, AK_2) = AM(K_1, K_2)A^T$. For the direction "$\subseteq$", let $Y \in M(AK_1, AK_2)$. Then by (c), for every $u \in K_1^*, v \in K_2^*$ we have that $u^T A^T Y A v \geq 0$. We therefore have that

$$\begin{pmatrix} Y & \mathbf{0} \\ \mathbf{0}^T & 0 \end{pmatrix} = A^T Y A \in M(K_1, K_2).$$

Multiplying by A from the left and by A^T from the right, we obtain (since AA^T is the identity matrix) that $Y \in AM(K_1, K_2)A^T$.

For the converse direction "$\supseteq$", let $Y \in AM(K_1, K_2)A^T$. Since $K_1 \subseteq \{x : x_n = 0\}$, every matrix in $M(K_1, K_2)$ has only zeros in row n, and by the symmetry (a) it has only zeros also in column n. Hence,

$$A^T Y A = \begin{pmatrix} Y & \mathbf{0} \\ \mathbf{0}^T & 0 \end{pmatrix} \in M(K_1, K_2).$$

[2] Note that the application of N in the right-hand side is in a smaller dimension than in the left-hand side.

By (c), for every $u \in K_1^*, v \in K_2^*$ it holds that $u^T A^T Y A v \geq 0$, and hence $Y \in M(AK_1, AK_2)$.

Now since $A^T e_0$ is just e_0 (in a larger dimension), we conclude that

$$N(AK_1, AK_2) = AM(K_1, K_2)A^T e_0 = AM(K_1, K_2)e_0 = AN(K_1, K_2).$$

The proof for the N_+ operator is similar since Y is positive semidefinite if and only if $A^T Y A$ is (observe that Y has a Gram matrix representation if and only if $A^T Y A$ has such a representation). □

LEMMA A.11. *Let $F = Q \cap \{x : \forall i \in I_0, x_i = 0\} \cap \{x : \forall i \in I_1, x_i = x_0\}$, and let f be the deletion operation of the coordinates $I_0 \cup I_1$. If $K_1 \subseteq F$ and $K_2 \subseteq Q$ are convex cones, then $f(N(K_1, K_2)) = N(f(K_1), f(K_2 \cap F))$, and similarly for N_+.*

Proof. K_1 and $K_2 \cap F$ are both contained in F, so we can repeatedly apply Lemma A.10 to them and delete the coordinates of $I_0 \cup I_1$. (Note that by using Lemma A.5 we can extend Lemma A.10 also to deleting coordinates that are fixed to x_0.) It follows that $f(N(K_1, K_2 \cap F)) = N(f(K_1), f(K_2 \cap F))$.

By Lemma A.6 we have that $N(K_1, K_2 \cap F) = N(K_1, K_2) \cap F$, and since $N(K_1, K_2) \subseteq K_1 \subseteq F$, we have that $N(K_1, K_2 \cap F) = N(K_1, K_2)$. The proof follows. □

COROLLARY A.12. *Let $F = Q \cap \{x : \forall i \in I_0, x_i = 0\} \cap \{x : \forall i \in I_1, x_i = x_0\}$, and let f be the deletion operation of the coordinate $I_0 \cup I_1$. If $K \subseteq F$ is a convex cone, then $f(N(K)) = N(f(K))$,[3] and similarly for N_+, N_{FR}, and $N_{\mathrm{FR}+}$.*

Proof. For the N operator we have from Lemma A.11 that

$$f(N(K)) = N(f(K), f(Q \cap F)),$$

and $f(Q \cap F)$ is just Q in the smaller dimension, so $f(N(K)) = N(f(K))$. The proof for the N_+ operator is similar.

For the N_{FR} operator we have from Lemma A.11 that

$$f(N_{\mathrm{FR}}(K)) = N(f(K), f(\mathrm{FR}(G) \cap F)),$$

and it follows from Lemmas A.8 and A.9 that $f(\mathrm{FR}(G) \cap F) = \mathrm{FR}(G - I_0 - I_1) \cap H$, where $H = \{x : x_i = 0 \ \forall i \in \Gamma(I_1) - I_0 - I_1\}$. We therefore have that

$$f(N_{\mathrm{FR}}(K)) = N(f(K), \mathrm{FR}(G - I_0 - I_1) \cap H).$$

Note that $f(K) \subset H$ since $K \subseteq F \cap \mathrm{FR}(G) \subseteq H$, and so by Lemma A.6 we have that $f(N_{\mathrm{FR}}(K)) = N_{\mathrm{FR}}(f(K))$, as required. The proof for $N_{\mathrm{FR}+}(K)$ is similar. □

Removing vertices from the graph. For the stable set problem, the properties collected so far, and in particular Corollary A.12, give a useful characterization to whether $x \in N^r(G)$ in the case that x has a fixed coordinate (i.e., $x_i = 0$ or $x_i = x_0$).

Recall that $V = \{1, \ldots, n\}$. For a vector $x \in \mathbb{R}^n$ and a subset $W \subset V$, we denote by x_W the restriction of x to the coordinates of W. We can now prove Lemma 2.5, showing that if $x \in \mathbb{R}^n$ with $x_i = 1$ and $x_j = 0$ for all $j \in \Gamma(i)$, then for all $r \geq 0$, $x \in N^r(G)$ if and only if $x_{V - \Gamma(i) - i} \in N^r(G - \Gamma(i) - i)$, and similarly for N_+^r, N_{FR}^r, and $N_{\mathrm{FR}+}^r$.

Proof of Lemma 2.5. It is clear that x belongs to the face F of Q that is defined by the hyperplanes $\{x : x_i = x_0\}$ and $\{x : x_j = 0\}$ for all $j \in \Gamma(i)$. Then $x \in N^r(G)$ if and only if $x \in N^r(G) \cap F$, which is equivalent, by Corollary A.7, to

[3] Note that the application of N in the right-hand side is in a smaller dimension than in the left-hand side.

$x \in N^r(\mathrm{FR}(G) \cap F)$. Let f be the deletion operation of the coordinates $\Gamma(i) \cup \{i\}$, and then we have equivalently that $f(x) \in f(N^r(\mathrm{FR}(G) \cap F))$. By Corollary A.12, the latter is equivalent to $f(x) \in N^r(f(\mathrm{FR}(G) \cap F))$. By Lemmas A.8 and A.9, we have that $f(\mathrm{FR}(G) \cap F) = \mathrm{FR}(G - \Gamma(i) - i)$, and the proof follows. The proof for N_+^r, N_{FR}^r, and $N_{\mathrm{FR}+}^r$ is similar. □

LEMMA A.13. *Let $x \in \mathbb{R}^n$ be a vector and assume that $x_i = 0$ for some i. Then $x \in N^r(G)$ if and only if $x_{V-i} \in N^r(G-i)$, and similarly for N_+^r, N_{FR}^r, and $N_{\mathrm{FR}+}^r$.*

Proof. It is clear that x belongs to the face F of Q that is defined by the hyperplane $x_i = 0$. Then $x \in N^r(G)$ if and only if $x \in N^r(G) \cap F$, which is equivalent, by Corollary A.7, to $x \in N^r(\mathrm{FR}(G) \cap F)$. Let f be the deletion operation of the coordinate i, and then we have equivalently that $f(x) \in f(N^r(\mathrm{FR}(G) \cap F))$. By Corollary A.12, the latter is equivalent to $f(x) \in N^r(f(\mathrm{FR}(G) \cap F))$. By Lemma A.8 we have that $f(\mathrm{FR}(G) \cap F) = \mathrm{FR}(G - i)$, and the proof follows. The proof for N_+^r, N_{FR}^r, and $N_{\mathrm{FR}+}^r$ is similar. □

A.2. Bounds on the rank. We describe some general methods to obtain upper and lower bounds on the N-rank and N_+-rank of valid inequalities and extend them to the N_{FR}-rank. We also illustrate the use of these methods on a few valid constraints for the stable set problem (see Table 1).

The N-rank of an inequality valid for $\mathrm{STAB}(G)$ depends only on the subgraph induced by those vertices with a nonzero coefficient, and similarly for N_+, N_{FR}, and $N_{\mathrm{FR}+}$. Indeed, if a vertex i has a zero coefficient, then the inequality being valid for $N^r(G)$ is equivalent, by Lemma 2.4, to the inequality being valid for $N^r(G) \cap \{x : x_i = 0\}$, which in turn is equivalent, by Lemma A.13, to the inequality being valid for $N^r(G - i)$.

Upper bounds on the N-rank. Lovász and Schrijver [23] give an upper bound on $N(K)$, which allows us to upper bound the N-rank of an inequality, as follows.

The *sum* of two sets $K', K'' \subseteq \mathbb{R}^{n+1}$ is defined as $K' + K'' := \{x' + x'' : x \in K', x'' \in K''\}$. Note that if K', K'' are convex cones in Q, then $K'+K''$ is also a convex cone in Q. Furthermore, if K', K'' are obtained via the homogenization procedure (1)–(2) from polytopes $P', P'' \subseteq \mathbb{R}^n$, respectively, then $K' + K''$ corresponds to all convex combinations of a point from P' and a point from P'' (recall that x_0 needs to be scaled to 1).

LEMMA A.14 (Lovász and Schrijver [23]). *For all $1 \le i \le n$,*

$$N(K) \subseteq \Big(K \cap \{x : x_i = 0\}\Big) + \Big(K \cap \{x : x_i = x_0\}\Big).$$

Proof. If $x \in N(K)$, then there exists $Y \in M(K)$ with $x = Ye_0 = Ye_i + Yf_i$ for any $i \le i \le n$. Clearly, $Ye_i \in K \cap \{x : x_i = x_0\}$ and $Yf_i \in K \cap \{x : x_i = 0\}$, and the proof follows. □

COROLLARY A.15. *If an inequality is valid for both $K \cap \{x : x_i = 0\}$ and $K \cap \{x : x_i = x_0\}$, then it is valid for $N(K)$.*

Goemans and Tunçel [10] note that repeatedly using Lemma A.14 and Corollary A.7 gives that, for all $I \subseteq \{1, \dots, n\}$ with $|I| = r$,

$$N^r(K) \subseteq \sum_{I_0 \subseteq I} \Big(K \cap \{x : \forall i \in I_0, x_i = 0\} \cap \{x : \forall i \in I \setminus I_0, x_i = x_0\}\Big).$$

In particular, this shows that the N-rank of any cone K is at most n, proving Theorem 2.1.

For the stable set problem, Corollary A.15 can be rephrased as follows (using Lemmas 2.5 and A.13).

LEMMA A.16 (Lovász and Schrijver [23]). *Let P be a convex set with* STAB $\subseteq P \subseteq$ FRAC. *If $a^T x \leq b$ is an inequality such that for some $i \in V$, both the deletion and contraction of i give an inequality valid for P, then $a^T x \leq b$ is valid for $N(P)$.*

For example, if C induces a chordless odd cycle in G, the *odd hole constraint*

$$\sum_{i \in C} x_i \leq \frac{|C| - 1}{2} \tag{22}$$

has N-rank at most (and actually exactly) 1, because both the contraction and the deletion of any vertex result in an inequality that is valid for FRAC. (In fact, Lovász and Schrijver [23] prove that N(FRAC) is exactly the relaxation that is obtained by adding to FRAC all the odd hole constraints.)

Lovász and Schrijver [23] also give the following upper bound on the N-rank of a graph. The proof follows by applying Lemma A.16 repeatedly for $n - \alpha(G) - 1$ vertices outside a maximum stable set in the graph, since the graph induced on the other vertices must be bipartite.

COROLLARY A.17 (Lovász and Schrijver [23]). *The N-rank of a graph G with stability number $\alpha(G)$ is at most $n - \alpha(G) - 1$.*

It follows that the N-rank of any graph G is at most $n-2$. Note that the N-rank of FR is at most $n - 2$, while the N-rank of a general cone K is at most (and can actually be) n.

We next analyze the N-rank of a few more examples, due to Lovász and Schrijver [23]. By Corollary A.17, if B is a clique in G, the *clique constraint*

$$\sum_{i \in B} x_i \leq 1 \tag{23}$$

has N-rank at most (and actually exactly) $|B| - 2$. Note that the class of all clique constraints strengthens the class of all edge constraints (5).

If D induces a chordless odd cycle in $\overline{G}$ (the edge complement of G), the *odd antihole constraint*

$$\sum_{i \in D} x_i \leq 2 \tag{24}$$

has N-rank at most (and actually exactly) $(|D| - 3)/2$, because the contraction of a vertex results in an inequality trivially valid for FRAC, and the deletion of a vertex results in an inequality that is the sum of two clique constraints, each of size $(|D|-1)/2$ and hence of N-rank $(|D| - 5)/2$.

If W induces an odd wheel in G with center $i_0 \in W$, the *odd wheel constraint*

$$\sum_{i \in W \setminus \{i_0\}} x_i + \frac{|W| - 2}{2} x_{i_0} \leq \frac{|W| - 2}{2} \tag{25}$$

has N-rank at most (and actually exactly) 2, since the contraction of the center vertex results in a trivial inequality, and the deletion of the center vertex results in the odd hole constraint.

Upper bounds on the $N_{\rm FR}$-rank. The methods for obtaining upper bounds on the N-rank can be extended (with modifications) to upper bounds on the $N_{\rm FR}$-rank, as follows.

LEMMA A.18. *For all $ij \in E$,*

$$N(K) \subseteq \Big(K \cap \{x : x_i = x_j = 0\}\Big) + \Big(K \cap \{x : x_j = x_0\}\Big) + \Big(K \cap \{x : x_i = x_0\}\Big).$$

Proof. If $x \in N_{\rm FR}(K)$, then there exists $Y \in M(K)$ with $x = Ye_0 = Ye_i + Ye_j + Yf_{ij}$ for any $ij \in E$. Clearly, $Ye_i \in K \cap \{x : x_i = x_0\}$, $Ye_j \in K \cap \{x : x_j = x_0\}$, and $Yf_{ij} \in K \cap \{x : x_i = x_j = 0\}$, and the proof follows. □

COROLLARY A.19. *Let $ij \in E$. If an inequality is valid for $K \cap \{x : x_i = x_0\}$, for $K \cap \{x : x_j = x_0\}$, and for $K \cap \{x : x_i = x_j = 0\}$, then it is valid for $N_{\rm FR}(K)$.*

Corollary A.19 can be rephrased as follows (using Lemmas 2.5 and A.13).

LEMMA A.20. *Let P be a convex set with* STAB $\subseteq P \subseteq$ FRAC. *If $a^T x \le b$ is an inequality such that for some $ij \in E$, the contraction of i, the contraction of j, and the deletion of $\{i, j\}$ give an inequality valid for P, then $a^T x \le b$ is valid for $N(P)$.*

The following upper bound on the $N_{\rm FR}$-rank of a graph follows by applying Lemma A.20 repeatedly on edges, so that the removal of their endpoints results in a bipartite graph (e.g., a matching that is maximal with respect to containment).

COROLLARY A.21. *Suppose that a graph G contains a set of β edges, the removal of whose endpoints results in a bipartite graph. Then the $N_{\rm FR}$-rank of G is at most β.*

It follows that the $N_{\rm FR}$-rank of a graph G is at most $(n-2)/2$ if n is even and $(n-1)/2$ if n is odd; in general it is at most $\lfloor (n-1)/2 \rfloor$. In particular, the $N_{\rm FR}$-rank of the clique constraint (23) is at most $\lfloor (|B|-1)/2 \rfloor$.

We can apply these bounds to the other examples. The $N_{\rm FR}$-rank of the odd hole constraint (22) is at most (and thus exactly) 1, since the $N_{\rm FR}$ operator is at least as strong as N. The $N_{\rm FR}$-rank of the odd antihole constraint (24) is at most $\lfloor (|D|+1)/4 \rfloor$, because the contraction of a vertex results in an inequality trivially valid for FRAC, and the deletion of two vertices results in an inequality that is the sum of two clique constraints, each of size at most $(|D|-1)/2$ and hence of $N_{\rm FR}$-rank $\lfloor (|D|-3)/4 \rfloor$.[4] The $N_{\rm FR}$-rank of the wheel constraint (25) is at most (and thus exactly) 1, since the contraction of the center vertex results in a trivial inequality, the contraction of a noncenter vertex results in an inequality that is valid for FRAC, and the deletion of these two vertices also results in an inequality that is valid for FRAC.

Lower bounds on the N-rank. Lovász and Schrijver [23] show that certain uniform fractional stable sets belong to $N^r(G)$, regardless of the graph G. For example, for $r = 0$ it is straightforward that $(1/2)\mathbf{1} \in$ FRAC(G). The following lemma allows us to extend this to larger r, with the uniform solution being smaller, depending on r.

LEMMA A.22 (Lovász and Schrijver [23]). *Assume that P is down-monotone and contains* STAB(G). *If $(1/r)\mathbf{1} \in P$ for $r > 0$, then $1/(r+1)\mathbf{1} \in N(P)$.*

Proof. Let K be the convex cone obtained from P via the homogenization procedure (1)–(2). Define the matrix $Y \in \mathbb{R}^{(n+1)\times(n+1)}$ by

$$Y_{ij} = \begin{cases} 1 & \text{if } i = j = 0, \\ 1/(r+1) & \text{if } (i = 0, j > 0) \text{ or } (i > 0, j = 0) \text{ or } (i = j > 0), \\ 0 & \text{otherwise.} \end{cases}$$

[4] In fact, direct calculations show that the $N_{\rm FR}$-rank of the odd antihole constraint (24) with $|D| = 7$ is at most 1.

To see that $Y \in M(K,Q)$ observe that (a), (b) clearly hold, and let us now show that (c$''$) holds:

$$Ye_i = \frac{1}{t+1}(e_0 + e_i) \in \mathrm{ST}(G) \subseteq K$$

and

$$Yf_i = \frac{r}{r+1}e_0 + \sum_{j\neq 0,i} \frac{1}{r+1}e_j = \frac{r}{r+1}\left(e_0 + \sum_{j\neq 0,i}\frac{1}{r}e_j\right).$$

By the induction hypothesis we have that

$$\sum_{j\neq 0,i}\frac{1}{r}e_j \leq \sum_{j\neq 0}\frac{1}{r}e_j \in P,$$

and the down-monotonicity of P implies that $Yf_i \in K$, and thus (c$''$) holds. We conclude that $Ye_0 \in N(K)$, i.e., $1/(r+1)\mathbf{1} \in N(P)$. □

COROLLARY A.23 (Lovász and Schrijver [23]). *$1/(r+2)\mathbf{1} \in N^r(G)$ for all $r \geq 0$.*

Proof. Proceed by induction on r. We mentioned above that the case $r = 0$ is trivial. The inductive step follows from Lemma A.22, since $N^r(\mathrm{FRAC}(G))$ clearly contains $\mathrm{STAB}(G)$ and is down-monotone by Lemma 2.4. □

COROLLARY A.24 (Lovász and Schrijver [23]). *The N-rank of a graph G with stability number $\alpha(G)$ is at least $n/\alpha(G) - 2$.*

Proof. Let r be the N-rank of G, and hence $N^r(G) = \mathrm{STAB}(G)$. By Corollary A.23 we have that $1/(r+2)\mathbf{1} \in N^r(G)$. The inequality $\mathbf{1}^T x \leq \alpha$ is valid for $\mathrm{STAB}(G) = N^r(G)$, and in particular for $1/(r+2)\mathbf{1}$, implying that $n/(r+2) \leq \alpha(G)$, and the proof follows. □

For example, the stability number of a clique B is 1, so the N-rank of B is at least, and hence exactly, $|B| - 2$. In fact, the above proof shows that the N-rank of the clique constraint (23) is at least, and hence exactly, $|B| - 2$. The stability number of an odd antihole D is 2, so the N-rank of D is at least $|D|/2 - 2$, and since $|D|$ is odd, it must be at least $(|D| - 3)/2$. In fact, this shows that the N-rank of the odd antihole constraint (24) is at least, and hence exactly, $(|D| - 3)/2$. Corollary A.23 also yields a lower bound on the N-rank of the wheel constraint (25). Indeed, let r be the N-rank of this constraint. Then we have that this constraint is valid for $N^r(G)$ and, in particular, for $1/(r+2)\mathbf{1} \in N^r(G)$. Thus,

$$\frac{1}{r+2}\left(|W| - 1 + \frac{|W|-2}{2}\right) \leq \frac{|W|-2}{2},$$

which gives us that $\frac{2(|W|-1)}{|W|-2} + 1 \leq r + 2$ and thus $r \geq 1 + \frac{2}{|W|-2}$. Since the N-rank of the wheel constraint is an integer, it must be at least, and hence exactly, 2.

Lower bounds on the N_{FR}-rank. The methods for obtaining lower bounds on the N-rank can be extended (with modifications) to lower bounds on the N_{FR}-rank, as follows.

LEMMA A.25. *Assume that P be down-monotone and contains $\mathrm{STAB}(G)$. If $(1/r)\mathbf{1} \in P$ for $r > 0$, then $1/(r+2)\mathbf{1} \in N_{\mathrm{FR}}(P)$.*

Proof. Define the matrix $Y \in \mathbb{R}^{(n+1)\times(n+1)}$ by

$$Y_{ij} = \begin{cases} 1 & \text{if } i = j = 0, \\ 1/(r+2) & \text{if } (i = 0, j > 0) \text{ or } (i > 0, j = 0) \text{ or } (i = j > 0), \\ 0 & \text{otherwise.} \end{cases}$$

To see that $Y \in M(K, FR)$ observe that (a), (b) clearly hold, and let us now show that (c$''$) holds:

$$Ye_i = \frac{1}{r+2}(e_0 + e_i) \in \mathrm{ST}(G) \subseteq K$$

and for $ij \in E$

$$Yf_{ij} = \frac{r}{r+2}e_0 + \sum_{l \neq 0,i,j} \frac{1}{r+2}e_l = \frac{r}{r+2}\left(e_0 + \sum_{l \neq 0,i,j} \frac{1}{r}e_l\right).$$

By the induction hypothesis we have that

$$\sum_{l \neq 0,i,j} \frac{1}{r}e_l \leq \sum_{l \neq 0} \frac{1}{r}e_l \in P,$$

and the down-monotonicity of P implies that $Yf_{ij} \in K$, and thus (c$''$) holds. We conclude that $Ye_0 \in N_{\mathrm{FR}}(K)$, i.e., $1/(r+2)\mathbf{1} \in N_{\mathrm{FR}}(P)$. □

COROLLARY A.26. *$1/(2r+2)\mathbf{1} \in N^r_{\mathrm{FR}}(G)$ for all $r \geq 0$.*

Proof. Proceed by induction on r. We mentioned above that the case $r = 0$ is trivial. The inductive step follows from Lemma A.25, since $N^r_{\mathrm{FR}}(\mathrm{FRAC}(G))$ clearly contains $\mathrm{STAB}(G)$ and is down-monotone by Lemma 2.4. □

COROLLARY A.27. *The N_{FR}-rank of a graph G with stability number $\alpha(G)$ is at least $n/(2\alpha(G)) - 1$.*

Proof. Let r be the N-rank of G, and hence $N^r(G) = \mathrm{STAB}(G)$. By Corollary A.26 we have that $1/(r+2)\mathbf{1} \in N^r(G)$. The inequality $\mathbf{1}^T x \leq \alpha(G)$ is valid for $\mathrm{STAB}(G) = N^r(G)$, and in particular for $1/(r+2)\mathbf{1}$, implying that $n/(2r+2) \leq \alpha(G)$, and the proof follows. □

For example, the N_{FR}-rank of a clique B is at least $|B|/2 - 1$ (since the stability number of B is 1), and it must be an integer, so we have that it is at least $\lfloor (|B|-1)/2 \rfloor$. In fact, the above proof shows that the N_{FR}-rank of the clique constraint (23) is at least, and hence exactly, $\lfloor (|B|-1)/2 \rfloor$. The N_{FR}-rank of an odd antihole D is at least $|D|/4 - 1$ (since the stability number of D is 2), and it must be an integer (while $|D|$ is odd), so we have that it is at least $\lfloor |D|/4 \rfloor$. In fact, this shows that the N-rank of the odd antihole constraint (24) is at least $\lfloor |D|/4 \rfloor$.

Upper bounds on the N_+-rank. Lovász and Schrijver [23] give also a sufficient condition for an inequality to be valid for $N_+(K)$. The following lemma considers an inequality $u^T x \geq 0$ with $u_0 \geq 0$ and $u_i \leq 0$ for $i \geq 1$. It can be extended to an arbitrary inequality $u^T x \geq 0$ by flipping the relevant coordinates according to Lemma A.5.

LEMMA A.28 (Lovász and Schrijver [23]). *If for all i with $u_i < 0$, $u^T x \geq 0$ is valid for $K \cap \{x : x_i = x_0\}$, then $u^T x \geq 0$ is valid for $N_+(K)$.*

By applying this to the stable set problem we obtain Lemma 2.6. Indeed, considering the original n-dimensional space, the inequalities $a^T x \leq b$ (with $a \in \mathbb{R}^n$) that are valid for $\mathrm{STAB}(G)$ are nontrivial only when $b > 0$ and $a \geq 0$, and then we can use Lemma A.28.

For example, the clique, odd hole, odd wheel, and odd antihole constraints all have N_+-rank at most (and thus exactly) 1. Lovász and Schrijver [23] show also that the so-called orthogonality constraints (see [21, 11] for a definition) are valid for $N_+(\mathrm{FRAC})$ by definition, and hence their N_+-rank is also 1.

One simple way to derive facet-defining valid inequalities from other facet-defining inequalities is *cloning* a clique at a vertex i. That is, replacing the vertex i by a clique and replacing every edge incident to i by corresponding edges that are incident to all the clique vertices and substituting the variable of i in the inequality with the sum of the variables of the clique vertices. In general, it is not clear how cloning influences the N_+-rank of an inequality. However, Goemans and Tunçel [10] note that Lemma 2.6 implies that cloning at the center vertex of an odd wheel inequality still has N_+-rank 1, and that cloning at one or several vertices of an odd wheel, odd hole, or odd antihole inequality has N_+-rank at most 2. Indeed, fixing any variable (of the corresponding subgraph) to 1, the resulting inequality can be seen to be a linear combination of clique inequalities and hence valid for $N_+(\text{FRAC})$.

COROLLARY A.29 (Lovász and Schrijver [23]). *If $G - \Gamma(i) - i$ has N_+-rank at most r for every $i \in V$, then the N_+-rank of G is at most $r + 1$.*

It follows, for example, that the N_+-rank of a clique, an odd antihole, or an odd wheel is at most (and hence exactly) 1. It also follows (as stated in Lemma 2.7) that the N_+-rank of a graph G is at most its stability number $\alpha(G)$. This bound is tight for a clique.

Lower bounds on the N_+-rank. Lovász and Schrijver [23] give no general method to lower bound the N_+-rank. The approach taken by Stephen and Tunçel [25], Goemans and Tunçel [10], and Cook and Dash [5] is to obtain an analogue of Corollary A.23 that holds for a specific cone K. That is, they show that $N_+^r(K)$ contains a "uniform" solution that does not belong to K_I, and thus obtain that the N_+-rank of K must be larger than r. Our analysis in section 3 also follows this approach.

We note that Goemans and Tunçel [10] give a sufficient condition for $N_+(K) = N(K)$ to hold, but this condition does not appear to be applicable to the stable set problem.

The ranks of the constraints exemplified above are listed in Table 1.

TABLE 1
The ranks of some example constraints.

Constraint	N-rank	N_{FR}-rank	N_+-rank	$N_{\text{FR}+}$-rank
odd hole (22)	1	1	1	1
clique (23)	$\lvert B\rvert - 2$	$\lfloor(\lvert B\rvert - 1)/2\rfloor$	1	1
antihole (24)	$(\lvert D\rvert - 3)/2$	$\lfloor\lvert D\rvert/4\rfloor \le \text{rank} \le \lfloor(\lvert D\rvert + 1)/4\rfloor$	1	1
wheel (25)	2	1	1	1

REFERENCES

[1] N. ALON, M. KRIVELEVICH, AND B. SUDAKOV, *Finding a large hidden clique in a random graph*, Random Structures Algorithms, 13 (1998), pp. 457–466.

[2] N. ALON, M. KRIVELEVICH, AND V. H. VU, *On the concentration of eigenvalues of random symmetric matrices*, Israel J. Math., to appear.

[3] N. ALON AND J. H. SPENCER, *The Probabilistic Method*, John Wiley, New York, 1992.

[4] R. BOPPANA AND M. M. HALLDÓRSSON, *Approximating maximum independent sets by excluding subgraphs*, BIT, 32 (1992), pp. 180–196.

[5] W. COOK AND S. DASH, *On the matrix-cut rank of polyhedra*, Math. Oper. Res., 26 (2001), pp. 19–30.

[6] U. FEIGE, *Randomized graph products, chromatic numbers, and the Lovász ϑ-function*, Combinatorica, 17 (1997), pp. 79–90.

[7] U. FEIGE AND R. KRAUTHGAMER, *Finding and certifying a large hidden clique in a semirandom graph*, Random Structures Algorithms, 16 (2000), pp. 195–208.

[8] A. Frieze and C. McDiarmid, *Algorithmic theory of random graphs*, Random Structures Algorithms, 10 (1997), pp. 5–42.
[9] Z. Füredi and J. Komlós, *The eigenvalues of random symmetric matrices*, Combinatorica, 1 (1981), pp. 233–241.
[10] M. X. Goemans and L. Tunçel, *When does the positive semidefiniteness constraint help in lifting procedures?*, Math. Oper. Res., 26 (2001), pp. 796–815.
[11] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, 2nd ed., Springer-Verlag, Berlin, 1993.
[12] J. Håstad, *Clique is hard to approximate within* $n^{1-\epsilon}$, in Proceedings of the 37th Annual Symposium on the Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 627–636.
[13] M. Jerrum, *Large cliques elude the Metropolis process*, Random Structures Algorithms, 3 (1992), pp. 347–359.
[14] A. Juels and M. Peinado, *Hiding cliques for cryptographic security*, Des. Codes Cryptogr., 20 (2000), pp. 269–280.
[15] F. Juhász, *The asymptotic behaviour of Lovász' θ function for random graphs*, Combinatorica, 2 (1982), pp. 153–155.
[16] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum, New York, 1972, pp. 85–103.
[17] R. M. Karp, *The probabilistic analysis of some combinatorial search algorithms*, in Algorithms and Complexity, Academic Press, New York, 1976, pp. 1–19.
[18] D. E. Knuth, *The sandwich theorem*, Electron. J. Combin., 1 (1994), Article 1 (electronic).
[19] M. Krivelevich and V. H. Vu, *Approximating the independence number and the chromatic number in expected polynomial time*, in Proceedings of the 27th International Colloquium on Automata, Languages and Programming, Springer-Verlag, New York, 2000, pp. 13–24.
[20] L. Kučera, *Expected complexity of graph partitioning problems*, Discrete Appl. Math., 57 (1995), pp. 193–212.
[21] L. Lovász, *On the Shannon capacity of a graph*, IEEE Trans. Inform. Theory, 25 (1979), pp. 1–7.
[22] L. Lovász, *Stable sets and polynomials*, Discrete Math., 124 (1994), pp. 137–153.
[23] L. Lovász and A. Schrijver, *Cones of matrices and set-functions and* 0-1 *optimization*, SIAM J. Optim., 1 (1991), pp. 166–190.
[24] F. McSherry, *Spectral partitioning of random graphs*, in Proceedings of the 42nd Annual IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 2001, pp. 529–534.
[25] T. Stephen and L. Tunçel, *On a representation of the matching polytope via semidefinite liftings*, Math. Oper. Res., 24 (1999), pp. 1–7.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 371–385

STABILITY OF ADAPTIVE AND NONADAPTIVE PACKET ROUTING POLICIES IN ADVERSARIAL QUEUEING NETWORKS*

DAVID GAMARNIK†

Abstract. We investigate the stability of packet routing policies in adversarial queueing networks. We provide a simple classification of networks which are stable under any greedy scheduling policy. We show that a network is stable if and only if the underlying undirected connected graph contains at most two edges. We also propose a simple and distributed policy which is stable in an arbitrary adversarial queueing network even for the critical value of the arrival rate $r = 1$. Finally, a simple and checkable network flow-type load condition is formulated for adaptive adversarial queueing networks, and a policy is proposed which achieves stability under this new load condition. This load condition is a relaxation of the integral network flow-type condition considered previously in the literature.

Key words. congestion, scheduling, multicommodity flow

AMS subject classifications. 68M20, 60K25, 90B10, 90B25

PII. S0097539700369168

1. Introduction.

The focus of this paper is the stability of adversarial queueing systems. Such queueing models have attracted much attention recently as a convenient tool for modeling packet injection and routing in a communication network. An adversarial assumption on the nature of the incoming traffic substitutes more traditional stochastic arrival assumptions. Two types of queueing networks are usually considered: circuit switch and packet switch networks (also referred to as adaptive and nonadaptive packet routing networks). In the first model, an adversary injects packets for processing, specifying the paths that the packets have to follow. The scheduler needs to decide which packets to process when several packets are competing for the same edge. Such models have been introduced by Borodin et al. in [6] and considered subsequently in several papers [3], [9], [11], [17].

In packet switch networks, an adversary injects packets and specifies only their origin and destination. The scheduler is free to choose a path along which the packets are processed. This model has been considered only recently by Aiello et al. [1]. In both models, the goal of the scheduler is to keep the queue lengths of packets competing for the same edge as small as possible. While constructing schedules which guarantee minimal queue length is a computationally intractable problem (even static circuit switch and packet switch scheduling problems are NP-complete), researchers have focused on schedules which at least guarantee bounded queue lengths at all times, i.e., stability.

1.1. Stability of nonadaptive packet routing schedules.

A natural necessary condition for stability exists in circuit switch-type networks. A positive integer w (called burstiness) exists such that, for any edge e and any time interval $[t_1, t_2)$, the total number of packets that are injected and contain edge e on their paths should not be bigger than $t_2 - t_1 + w$ (assuming each edge processes packets with unit speed).

*Received by the editors March 13, 2000; accepted for publication (in revised form) July 30, 2002; published electronically January 28, 2003. A preliminary version of this paper appeared in *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, ACM, New York, 1999, pp. 206–214.
http://www.siam.org/journals/sicomp/32-2/36916.html
†T.J. Watson Research Center, IBM, Yorktown Heights, NY 10598 (gamarnik@watson.ibm.com).

If the load condition is systematically violated, the queue lengths will build up no matter what schedule is used. The focus of the research has been understanding when the load condition is also sufficient for stability. It was proven that acyclic and unidirectional ring queueing networks are stable whenever the load condition is met and an arbitrary greedy schedule is implemented [3], [6], [16]. Meanwhile, certain natural policies were shown to be unstable even if the load condition holds. Andrews [2] and Andrews et al. [3] showed that First-In-First-Out (FIFO) and Nearest-To-Go (NTG) policies can be unstable. The instability of FIFO policies was also shown before by Bramson [7] for nonadversarial (stochastic) queueing networks. Borodin et al. [6] showed that NTG policy can be unstable in certain networks even if the arrival rate of the packets is bounded by an arbitrarily small constant r. On the other hand, Furthest-To-Go policy is stable in all networks (Andrews et al. [3]) whenever the maximal arrival rate r is strictly smaller than one. Goel [11] provided a complete algorithmic characterization of directed graphs which are stable for all greedy scheduling rules. Such characterization can be adapted to undirected graphs in which packets competing for an edge from opposite sides can simultaneously cross the edge. Our assumption throughout the paper will be that only one packet can cross any given edge at a time from either end, and all the graphs are assumed to be undirected. Obtaining a complete algorithmic classification of stable networks for every policy seems to be an unachievable task. It is shown in Gamarnik [10] that checking stability for a class of *generalized priority* policies is an algorithmically undecidable problem. Whether undecidability holds for more common policies like FIFO or priority policies remains to be seen.

1.2. The stability of adaptive packet routing schedules. The stability of adaptive packet routing schedules in adversarial queueing networks has only recently been analyzed by Aiello et al. [1]. This model does not have a natural load condition for stability, as opposed to nonadaptive queueing models. There is no explicit load on edges implied by the incoming traffic; rather, the load depends on the routing policy used. Aiello et al. thus introduced the following assumption. Suppose, for some positive integer w and some positive real $r < 1$, that an adversary can associate with each incoming packet a path in such a way that, in every time interval $[t, t+w)$, every edge has been assigned to no more than rw paths. In other words the adversary should be able to reformulate the problem in the nonadaptive sense, described above, without revealing the underlying assigned paths. It was shown in [1] that a stable distributed routing policy exists under the assumption above. Also, the schedule does not assume the knowledge of the arrival rate r and interval w, and the total number of packets in the network is bounded by $O(m^{5/2}n^{5/2}w/(1-r))$, where m and n are the numbers of edges and nodes in the graph, respectively.

1.3. Results. A number of questions remain outstanding, some of which are listed in Borodin et al. [6]. It is not clear whether a ring-type queueing network allowing traffic in both directions is stable under any greedy scheduling rule. More generally, which networks are stable under all greedy scheduling rules (universally stable)? This question was resolved by Goel [11] for directed graph queueing networks but remains outstanding for undirected graphs in which each edge can be crossed by only one packet at a time from either end. We provide in this paper a very simple answer to this problem. A connected undirected graph is universally stable if and only if it contains at most two edges. We establish this result by proving universal stability for a simple graph with two edges, G_1, and constructing unstable greedy schedules for graphs G_2, G_3, G_4 (see Figure 1). We will show specifically that such an

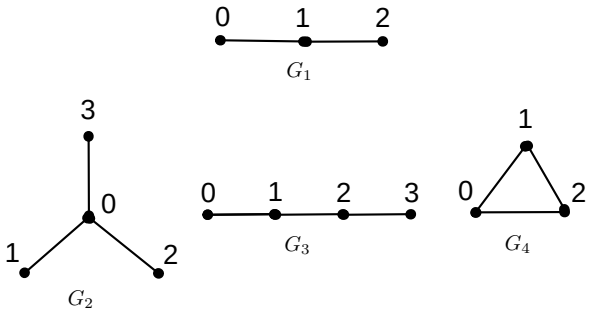


FIG. 1. *Graphs* G_1, G_2, G_3, G_4.

unstable greedy policy exists when the maximal arrival rate r satisfies $r^3 + r^4 > 1$ for the graph G_2 and satisfies $r^4 + r^5 > 1$ for the graphs G_3, G_4.

These schedules are very similar to the ones constructed by Goel [11] and Andrews et al. [3]. Clearly, any connected undirected graph with more than two edges contains one of the graphs G_2, G_3, G_4 as a subgraph and as a result is unstable. In particular, the ring-type queueing network either is graph G_4 or contains G_3 as a subgraph and, as a result, is not universally stable. We then propose a very simple distributed Nearest-To-Origin (NTO) priority policy and prove that this policy is stable in all adversarial queueing networks even for the critical arrival rate $r = 1$. This answers positively the question posed in [6] on the existence of a stable scheduling policy under a critical arrival rate $r = 1$.

For the case of adaptive packet routing models, we consider a more relaxed load condition than the one used in [1]. We assume that, for the packets that arrive during any time interval of the length w, the corresponding static multicommodity flow problem has a feasible *fractional* solution with maximal congestion (to be defined) not bigger than rw, where $r < 1$. That is, we relax the integrality requirement in the multicommodity flow-type constraint on the arriving traffic considered in [1]. We construct a simple discrete review-type policy based on the static packet routing problem and prove that this policy is stable under this relaxed load condition. The algorithm is based on an algorithm proposed in [5] for the static packet routing problem, which achieves asymptotic optimality as the network load diverges to infinity.

The advantage of the relaxed assumption above is clear—the load condition can be checked efficiently by solving a corresponding fractional multicommodity flow problem, whereas the integral multicommodity flow problem is NP-complete. Specifically, if the maximal arrival rate for any pair of origin-destination nodes (i, j) is r_{ij}, then there exists a stable packet routing protocol if the fractional multicommodity flow problem with parameters r_{ij} has a feasible solution with maximal congestion smaller than one. Our scheduling rule, however, would not assume the knowledge of the rates r_{ij}. We will also show that if, for any feasible solution, the maximal congestion is bigger than one, then no stable policy exists. We do not know whether a stable policy exists when the smallest maximal congestion is equal to one.

The disadvantage of our schedule compared to the one of Aiello et al. is that it occasionally needs information about the queue lengths in the entire network and thus is not distributed. Our bounds on maximal queue lengths are also inferior to the ones in [1].

We conclude with some open questions and directions for further research.

2. Definitions and assumptions. A nonadaptive adversarial queueing network is given as a graph (V, E). An adversary injects packets for processing. Each arriving packet has a prespecified path it has to follow. Once the end of the path is reached, the packet leaves the network. Each packet takes a unit time to cross a single edge, and only one packet in either direction can cross any given edge at a time. Packet processing occurs at integer time epochs $t = 0, 1, 2, \ldots$, although packet arrival can occur at an arbitrary real time. Packets that wait to cross some edge e accumulate into a queue at the vertex of e until chosen to cross. We introduce some additional notation in order to formally describe the dynamics. Let $\mathcal{P}$ be the set of all simple paths in the network (the set of paths that can be requested by packets).

For each path $P \in \mathcal{P}$, let $\{e_0^P, e_1^P, \ldots, e_{k(P)}^P\}$ be the set of consecutive edges in P. Let $A_P(t_1, t_2)$ denote the total number of packets injected during the time interval $[t_1, t_2)$ that request path P. For each $P \in \mathcal{P}$ and $e \in P$, let $D_{(e,P)}(t_1, t_2)$ be the total number of packets following path P that crossed edge e during the time interval $[t_1, t_2)$. In particular, $D_{(e,P)}(t, t+1)$ takes value 0 or 1 for each $t = 0, 1, 2, \ldots$. The values of $D_{(e,P)}(t, t+1)$ depend on the rule by which packets competing for the same edge are prioritized—the scheduling rule. Some examples of scheduling rules include First-In-First-Out (FIFO), in which packets are prioritized according to their arrival time into edge e, Longest-In-System, in which packets are prioritized according to their arrival time into the network, Shortest-In-System, Furthest-To-Go (FTG), and many others. Note that, whichever policy is used, the following restriction applies. For any edge e and time t,

$$\sum_{P:e\in P} D_{(e,P)}(t, t+1) \le 1.$$

In other words, at most one packet can cross an edge e during the time interval $[t, t+1)$. Finally, let $A_P(t) = A_P(0, t)$ and $D_{(e,P)}(t) = D_{(e,P)}(0, t)$, and let $Q_{(e,P)}(t)$ be the total number of packets following path P that are waiting to cross edge e at time t. The dynamics of the network is described as follows. For each $t = 0, 1, 2, \ldots$ and each path $P \in \mathcal{P}$,

$$Q_{(e_0^P,P)}(t) = Q_{(e_0^P,P)}(0) + A_P(t) - D_{(e_0^P,P)}(t) \tag{1}$$

and

$$Q_{(e_i^P,P)}(t) = Q_{(e_i^P,P)}(0) + D_{(e_{i-1}^P,P)}(t) - D_{(e_i^P,P)}(t) \tag{2}$$

for all $i = 1, 2, \ldots, k(P)$. We let $Q(t) = \sum_{e,P} Q_{(e,P)}(t)$ denote the total number of packets in the network at time t.

The packets are injected into the system by an adversary in a restricted manner. There exist a positive real number r, called the arrival rate, and a positive integer w with the following property. For each edge e, the total number of packets, injected during any interval $[t_1, t_2)$, whose assigned paths contain e is at most $r(t_2 - t_1) + w$. Formally, for each $e \in E$ and $t_1 < t_2$,

$$\sum_{P:e\in P} A_P(t_1, t_2) \le r(t_2 - t_1) + w. \tag{3}$$

This is the load assumption considered in [3], [6], [9], [11] and is a generalization of the path-specific arrival rate assumption considered earlier by Cruz [8].

The goal of the stability analysis is to understand the conditions under which the total number of packets in the network stays bounded, i.e., the conditions under which the network is stable. Specifically, we are interested in when a particular scheduling policy is stable and which networks are stable under an arbitrary greedy scheduling policy.

DEFINITION 1. *A scheduling policy in an adversarial queueing network* (V, E, r, w) *is defined to be greedy if, whenever there is a positive number of packets waiting to cross any given edge* e *at time* t*, at least one of these packets will cross* e *during interval* $[t, t+1)$*. Formally, for each* $e \in E$ *and* $t = 0, 1, 2, \ldots,$

$$\sum_{P:e\in P} Q_{(e,P)}(t) > 0$$

implies

$$\sum_{P:e\in P} D_{(e,P)}(t, t+1) = 1. \tag{4}$$

DEFINITION 2. *A scheduling policy in an adversarial queueing network* (V, E, r, w) *is defined to be stable if, under this policy, the total number of packets in the network stays bounded for all times. Namely,*

$$\sup_{t\in\Re_+} Q(t) < \infty.$$

A scheduling policy is defined to be universally stable if it is stable in all graphs. A (directed or undirected) graph (V, E) *is defined to be universally stable if every greedy policy in it is stable for all* $r < 1$ *and all nonnegative* w*.*

The necessary condition for stability is

$$r \le 1. \tag{5}$$

If this condition is violated, then an adversary can inject packets so that no scheduling rule will be able to keep the number of packets bounded.

An adaptive packet routing model is similar to the model above. An undirected graph (V, E) is given. An adversary injects packets but now specifies only their origin-destination pair $(i, j) \in V^2$. The goal of the scheduler is to select the paths for packets as well as to prioritize packets competing for the same edge. The total number of packets in the network again needs to be bounded. Immediately the following question arises: What is the analogue of the condition (5)? Aiello et al. [1] considered the following condition. A certain integer w and a real value $r < 1$ are fixed. It is assumed that the packets that arrived during any time interval $[t, t+w)$ can be associated with paths in the graph (V, E) in such a way that any edge e belongs to no more than rw paths. This condition can be reformulated using the integral multicommodity flow problem as follows. For a given graph (V, E) and a set of positive integers n_{ij}, $i, j \in V$, consider the following integer programming problem (we represent edges as pairs of nodes $(k, l) \in E$):

$$\begin{aligned} &\text{Minimize } C_{\max} \\ &\text{Subject to} \\ &\sum_{k:(i,k)\in E} x_{ik}^{ij} = n_{ij}, \qquad (i, j) \in V^2, \end{aligned} \tag{6}$$

$$\sum_{k:(k,j)\in E} x_{kj}^{ij} = n_{ij}, \qquad (i,j) \in V^2, \tag{7}$$

$$\sum_{l:(l,k)\in E} x_{lk}^{ij} = \sum_{l:(k,l)\in E} x_{kl}^{ij}, \ (i,j) \in V^2, \ k \neq i,j, \tag{8}$$

$$C_{kl} = \sum_{(i,j)\in V^2} x_{kl}^{ij}, \qquad (k,l) \in E, \tag{9}$$

$$C_{kl} \leq C_{\max}, \qquad (k,l) \in E, \tag{10}$$

$$x_{kl}^{ij}, C_{kl} \geq 0, \tag{11}$$

$$x_{ij}^{kl} \in Z_+. \tag{12}$$

Here x_{kl}^{ij} represents the number of packets going from node i to node j that pass through the edge (k,l). Equations (6)–(8) represent the conservation of flow. C_{kl} represents the total amount of integral flow assigned to any edge $(k,l) \in E$, and $C_{\max}$ represents the maximal amount of flow assigned to any edge $e \in E$.

It is not hard to prove that the load condition considered by Aiello et al. is equivalent to the following condition. Let $A_{ij}(t, t+w)$ denote the number of packets that arrived during the time interval $[t, t+w)$ and have an origin-destination pair (i,j). The condition is that, for any time t, the integral multicommodity flow problem above with input $n_{ij} = A_{ij}(t, t+w)$ has a solution $C_{\max}$ satisfying

$$C_{\max} \leq rw. \tag{13}$$

DEFINITION 3. *An adversarial queueing network is said to be of the type* (r, w, IMF) *(IMF stands for integral multicommodity flow) if the condition* (13) *is satisfied for any time* t*, where* $C_{\max}$ *is the optimal value of the integral multicommodity flow problem* (6)–(12) *on the input* $n_{ij} = A_{ij}(t, t+w)$*.*

An algorithm was constructed in [1] which achieves stability under the load condition (r, w, IMF) for networks of type (r, w, IMF) with $r < 1$. In this paper, we consider queueing networks of the type (r, w, FMF) (FMF stands for fractional multicommodity flow), where the load condition above is still assumed to be satisfied, but the solution to the multicommodity flow problem above need not be integral (constraint (12) is removed).

DEFINITION 4. *An adversarial queueing network is said to be of the type* (r, w, FMF) *if the condition* (13) *is satisfied for any time* t*, where* $C_{\max}$ *is the optimal value to the fractional multicommodity flow problem* (6)–(11) *on the input* $n_{ij} = A_{ij}(t, t+w)$*.*

Clearly our load condition is weaker. Since it uses a linear programming formulation, the condition is also efficiently checkable. We construct in section 4 a stable scheduling policy under this relaxed load condition whenever $r < 1$.

3. Universally stable graphs and universally stable policies. In the first part of this section, we focus on universal stability of undirected graphs. An exact characterization of directed stable graphs is given in [11]. Two directed graphs were constructed which are not universally stable. It is then proven that a directed graph is universally stable if and only if it does not contain one of these two graphs as a minor (for a definition of a graph minor, see [14]). This leads to an efficient algorithm for checking whether a given graph is stable.

In this section, we show that, for undirected graphs, the classification is even simpler. A graph with one edge and a graph G_1 with two edges are the only connected

undirected universally stable graphs. Note that the stability of unconnected graphs can be resolved by considering their connected components.

THEOREM 1. *A connected undirected graph is universally stable if and only if it has at most two edges.*

Remark. We conjecture that the graph with two edges is stable also for the critical arrival rate $r = 1$, but we do not have a proof.

Proof. We omit a trivial case of a graph with only one edge. We prove the stability of the graph G_1 by a simple reduction to a unidirectional ring network with two nodes. An adversarial queueing network (V, E, r, w) is called a unidirectional ring network if it is of the form $V = \{v_1, v_2, \ldots, v_n\}$, $E = \{e_1, e_2, \ldots, e_n\}$, $e_k = (v_k, v_{k+1})$, $k = 1, 2, \ldots, n-1$, $e_n = (v_n, v_1)$, and if every path P contains edges in increasing order modulo n. Namely, $P = \{e_j, e_{j+1}, \ldots, e_{j+k(P)}\}$ for some e_j, where the convention is to identify edge e_{n+i} with e_i. Thus a unidirectional ring is a circular form graph with all the packets moving in one direction. It was shown in [3] that the unidirectional ring is universally stable for all $r < 1$ and the total number of packets in the network at any moment is not bigger than $n^2 w/(1-r)$ (assuming initially that there are no packets in the network). We now show that, from the stability point of view, our graph G_1 is equivalent to the unidirectional ring with two nodes $V = \{v_1, v_2\}$ and two edges $E = \{e_{\text{lower}}, e_{\text{upper}}\}$ connecting nodes v_1 and v_2. Any packet in G_1 going from 0 to 1 or from 1 to 0 we associate with a packet going along the edge e_{lower} in the directions $v_1 \to v_2$. Any packet in G_1 going from 1 to 2 or from 2 to 1 we associate with a packet going along the edge e_{upper} in the directions $v_2 \to v_1$. We associate packets going along nodes $0, 1, 2$ or $2, 1, 0$ similarly. It is easy to see that this correspondence makes the two systems equivalent. In particular, graph G_1 is stable, and, if the initial number of packets is zero, then the maximal number of packets at any time is not more than $4w/(1-r)$.

We now prove the second part of the theorem. We will show that in any connected graph with more than two edges there exists an unstable greedy scheduling policy whenever $r > .86$. Clearly it suffices to prove the existence of such policies only for graphs G_2,G_3, G_4 on Figure 1.

Consider the graph G_2 first. The arrival pattern and the scheduling policy are described in several stages. Suppose initially that there are c packets waiting to cross the edge $(1, 0)$, where c is a sufficiently large number. During the time interval $[0, c)$, we process these c packets and generate rc packets requesting the path $1, 0, 2$. These packets do not move until time c. During the time interval $[c, c+rc)$, we process these rc packets and generate r^2c packets requesting path $2, 0, 3$ and r^2c packets requesting $0, 1$. These packets also do not move until the time $c + rc$. During the next time interval of the length r^2c, we generate r^3c packets requesting $1, 0, 3$ and r^3c packets requesting $2, 0$. The latter packets are processed before the previously generated $2, 0, 3$ packets. As a result, at time $c + rc + r^2c$, we obtain r^3c packets requesting $1, 0, 3$ and r^3c packets requesting $2, 0, 3$ (the latter generated in the previous round). During the next r^3c time units, we process entirely packets on the path $1, 0, 3$, process packets on the path $2, 0, 3$ along the edge $(2, 0)$, and generate r^4c packets requesting path $0, 3$. In the end, we obtain $r^3c + r^4c$ packets requesting edge $(0, 3)$. If $r^3 + r^4 > 1$ (which holds for $r > .82$), we end up with more than c packets requesting the path $0, 3$. Repeating the schedule for $c' = (r^3 + r^4)c$ packets starting from the edge $(0, 3)$, we obtain an unstable schedule. This completes the proof for the graph G_2.

The proof for the graph G_3 is very similar. Suppose initially that we have c packets requesting path $2, 1$. Process these packets, and generate rc packets requesting $2, 0$ during the time interval $[0, c)$. Process the new packets, and generate r^2c packets

requesting paths $1,2,3$ and $0,1$. Process these packets, and generate r^3c packets requesting $3,2,1$ and $0,1$. During the next r^3c time units, process all $0,1$ packets, and generate r^4c packets requesting $0,1,2$. Also during this time interval, generate and process r^4c packets requesting $3,2$, give them priority over previously generated $3,2,1$ packets, and in the remaining time process these $3,2,1$ packets. We obtain in the end r^4c packets requesting $0,1,2$ and r^4c packets requesting $3,2,1$. Note that all these packets require edge $(1,2)$. Process all the $0,1,2$ packets, process $3,2,1$ packets through their first edge $(3,2)$, and generate r^5c packets requesting $2,1$. As a result, we obtain $r^4c + r^5c$ packets requesting edge $2,1$. If $r^4 + r^5 > 1$, that is, $r > .86$, then we end up with more than c packets requesting edge $2,1$. It follows that the scheduling rule is unstable. The construction of the unstable policy in the graph G_4 is identical to the one of G_3, where we identify node v_3 of G_3 with v_0 of G_4. This completes the proof of the theorem. □

In the remainder of this section, we address the question of universal stability of specific policies. We propose a simple NTO policy and prove that it is stable in all graphs even for the critical arrival rate $r = 1$. The NTO policy gives priority to packets which have crossed the smallest amount of edges. Namely, if two packets following paths $P, P' \in \mathcal{P}$ compete for the same edge $e = e_i^P = e_j^{P'}$ and $i < j$, then the packet following P should be processed first. If $i = j$, then the packets are prioritized arbitrarily.

THEOREM 2. *NTO policy is stable in any network for $r = 1$.*

Proof. Let $Q_e(0)$ denote the total initial number of packets waiting to cross an edge e. Also let $Q_k(t)$ denote the total number of packets at time t which are within exactly k steps from the origin. That is,

$$Q_k(t) = \sum_{P \in \mathcal{P}} Q_{(e_k^P, P)}(t).$$

Let us call these packets layer k packets. We will show by induction by k that $Q_k(t)$ is bounded by a constant for all t.

Base step $k = 0$. Fix an edge e and a time t. Let $t_0 \in [0,t]$ be the largest time at which no packets of layer 0 (packets that have not crossed any edge yet) were waiting at e. If no such time exists, set $t_0 = 0$. The total number of layer 0 packets in e at time t_0 is then at most $Q_e(0)$. During the time interval $[t_0, t)$, at most $r(t - t_0) + w = t - t_0 + w$ layer 0 (external) packets that want to cross e have arrived. Also, by the choice of t_0, edge e was processing packets constantly during the time interval $[t_0, t)$. Since NTO policy is used, layer 0 packets have priority over all other packets. It follows that total number of layer 0 packets at the edge e at time t satisfies

$$\sum_{P: e_0^P = e} Q_{(e_0^P, P)}(t) \le Q_e(0) + t - t_0 + w - (t - t_0) = Q_e(0) + w.$$

As a result, $Q_0(t) \le \sum_e Q_e(0) + w|E|$ for all t. We denote $\sum_e Q_e(0) + w|E|$ by B_0.

Induction step. Suppose, for some constants B_j, $j = 0, 1, \ldots, k-1$, $Q_j(t) \le B_j$ for all $j \le k-1$ and for all times t. We will show that, for some constant B_k, $Q_k(t) \le B_k$ for all t. Again, fix an edge e and an arbitrary time t. Again let $t_0 \le t$ denote the largest time instance such that there were no layer k packets waiting to cross e. Then edge e was always processing packets during the time interval $[t_0, t)$ (packets in layer k or lower). The layer k packets that wait to cross e at time t then are composed only of packets which were in layers $j \le k-1$ at time t_0 or packets that

arrived externally during the time interval $[t_0, t)$ and have edge e as their kth edge on the requested path. The first group of packets has a size bounded by $\sum_{j=0}^{k-1} B_j$, by the induction assumption. The second is bounded by $\sum_{P:e_k^P=e} A_P(t_0, t)$. We now estimate the number of layer k packets that crossed e during the time interval $[t_0, t)$. Since NTO policy is used, these packets were not processed only when there were packets in layers up to $k-1$ that wanted to cross e. The number of such packets is bounded by $\sum_{j=0}^{k-1} B_j$, i.e., the total possible number of packets in layers up to $k-1$ at time t_0, plus $\sum_{j=0}^{k-1} \sum_{P:e_j^P=e} A_P(t_0, t)$, which is the number of new packets that arrived in $[t_0, t)$ and cross e within $k-1$ steps. We conclude that at least

$$\max\left\{0, t - t_0 - \sum_{j=0}^{k-1} B_j - \sum_{j=0}^{k-1} \sum_{P:e_j^P=e} A_P(t_0, t)\right\}$$

packets of layer k crossed e during the time interval $[t_0, t)$. We obtain

$$\sum_{P:e_k^P=e} Q_k(t) \le \sum_{j=0}^{k-1} B_j + \sum_{P:e_k^P=e} A_P(t_0, t) - \left(t - t_0 - \sum_{j=0}^{k-1} B_j - \sum_{j=0}^{k-1} \sum_{P:e_j^P=e} A_P(t_0, t)\right)$$

$$\le 2\sum_{j=0}^{k-1} B_j + w,$$

where the last inequality follows from (3) and $r = 1$. Thus the total number of layer k packets is bounded by $B_k = 2|E|(\sum_{j=0}^{k-1} B_j) + w|E|$. This completes the induction step. □

After this paper was written, it was pointed out to the author by Kleinberg [12] that a similar analysis shows the stability of FTG policy (which gives priority to packets closest to their destination) when $r = 1$. During the course of the proof, we obtained the following bound on the total number of packets in the network at time t:

$$|Q(t)| \le (2|E|)^{p_{\max}+1}(B_0 + w),$$

where B_0 is the initial number of packets in the network and $p_{\max}$ is the maximal length $|P|$ of paths $P \in \mathcal{P}$. Unfortunately, the bound is exponential in the network parameters. It is shown in [4] that both NTO and FTG lead to exponentially large queue sizes in certain networks. It is also shown that no bound better than $2^{\sqrt{\max|V|,|E|}}$ is possible for a whole class of distributed deterministic policies including NTG and FTG.

Note that, unlike the $r < 1$ case, stability under the $r = 1$ condition does not necessarily imply that all the packets are delivered within a finite time. Indeed, consider the graph G_1 operating under NTG policy. Assume that initially there are several packets requesting path $0, 1, 2$. Assume that at each integer moment $t = 1, 2, \ldots$ an adversary injects a packet following $1, 2$. These packets have priority over the initial packets and block them from processing forever. Thus the delivery time for the initial packets is infinity. Whether there exists a policy with bounded delivery time for every packet when $r = 1$ is an open question.

4. Stable scheduling policies in adaptive adversarial queueing networks. In this section, we focus on adaptive packet routing policies in adversarial queueing networks. We have an undirected graph (V, E) and parameters $r, w > 0$. An adversary generates packets and specifies only their origin-destination. The network flow load assumption (r, w, FMF), described in section 2, is assumed to be satisfied by an adversary traffic. The goal is to construct a policy which is stable under the (r, w, FMF) assumption, when $r < 1$, and show that no stable policy exists against any adversary traffic when $r > 1$. For the case in which $r < 1$, we consider a corresponding static packet routing problem on a fixed input n_{ij}, which is formulated as follows. Suppose, for each $i, j \in V$, that we are given n_{ij} packets which are required to go from node i to node j via some path selected by the scheduler. Each edge can process only one packet at one time unit. The objective is to find a routing schedule which would minimize the time until all the packets reach their destination (makespan time). This static version of the packet routing problem with the makespan objective has been considered before by Srinivasan and Teo [15] and Bertsimas and Gamarnik [5]. We use here an asymptotically optimal scheduling algorithm developed in [5] (the Packet Routing Synchronization Algorithm or PRSA) for this static packet routing problem. The following result was proven in [5].

THEOREM 3. *Let n_{ij} denote the number of packets that are present in the network at time 0 and have nodes $i, j \in V$ as their origin-destination pair. Let $C_{\max}$ be the optimal solution to the (fractional) multicommodity problem with input n_{ij}, $i, j \in V$. Then there exists a packet routing scheduling algorithm which brings all the packets to their destination (has makespan time) in not more than*

$$C_{\max} + O(|V|^3|E|\sqrt{C_{\max}}) \tag{14}$$

time units. Moreover, the algorithm is such that, after time $T = C_{\max} + |V|\sqrt{C_{\max}}$, not more than $2|V||E|\sqrt{C_{\max}} + |V|^2|E|$ packets are still present in the network.

Since $C_{\max}$ is a lower bound on any feasible makespan time, the schedule is asymptotically optimal when the total initial number of packets $\sum_{ij} n_{ij}$ diverges to infinity. We now use this scheduling algorithm to construct a stable policy in a network (V, E) which satisfies the load condition (r, w, FMF). The routing policy is of *Discrete Review* type and is described as follows. We assume that $T_0 = 0$. The number of packets of type (i, j) at time t is denoted by $Q_{ij}(t)$.

Discrete Review Algorithm. For $k = 0, 1, 2, \ldots$, let $C^k_{\max}$ denote the optimal value to the multicommodity flow problem with the input $n_{ij} = Q_{ij}(T_k)$. Set $T_{k+1} = T_k + C^k_{\max} + |V|\sqrt{C^k_{\max}}$. Implement the PRSA at time T_k to the input $n_{ij} = Q_{ij}(T_k)$ for the first $C^k_{\max} + |V|\sqrt{C^k_{\max}}$ time units, ignoring packets arriving after time T_k.

In other words, the Discrete Review Algorithm looks at times T_k, $k = 0, 1, 2, \ldots$, at the entire network and solves the corresponding static packet routing problem, ignoring packets that arrive after T_k. Intuitively, since the PRSA has makespan time close to the maximal congestion when the loads are high, then, by the (r, w, FMF) assumption, $C^{k+1}_{\max}/C^k_{\max} \approx r < 1$, and the Discrete Review policy is stable. The following result is obtained.

THEOREM 4. *Suppose an adversarial queueing network (V, E) satisfies the (r, w, FMF) load condition. If $r < 1$, then the Discrete Review routing schedule is stable. Moreover,*

$$\limsup_{t\to\infty} \sum_{ij} Q_{ij}(t) = O\left(\frac{|V|^4|E|^3 + w^2|E|}{(1-r)^2}\right). \tag{15}$$

If $r > 1$, then no stable routing policy exists.

Proof. First we prove the stability of the Discrete Review routing schedule. We show that, for any k,

$$C_{\max}^{k+1} \le r(C_{\max}^{k} + |V|\sqrt{C_{\max}^{k}}) + w + |V|^2|E|\sqrt{C_{\max}^{k}}. \tag{16}$$

In fact, the total number of type (i,j) packets in the network at time T_{k+1} consists of two types of packets. We have $A_{ij}(T_k, T_{k+1})$ packets that arrived during the time interval $[T_k, T_{k+1})$ and packets that arrived before time T_k and still have not been processed. Denote the latter packets by $\hat{Q}_{ij}(T_k)$. From Theorem 3, since $T_{k+1} - T_k = C_{\max}^{k} + |V|\sqrt{C_{\max}^{k}}$, we have

$$\sum_{ij} \hat{Q}_{ij}(T_k) \le 2|V||E|\sqrt{C_{\max}^{k}} + |V|^2|E| \le |V|^2|E|\sqrt{C_{\max}^{k}} \tag{17}$$

(as long as $|V|, C_{\max} \ge 4$, which we assume to hold). Since our network is of (r, w, FMF) type, the optimal value of the multicommodity flow problem on the input $n_{ij} = A_{ij}(T_k, T_{k+1})$ is at most

$$r\left(\left\lceil \frac{T_{k+1} - T_k}{w} \right\rceil w\right) \le r(T_{k+1} - T_k) + w.$$

The optimal value of the multicommodity flow problem on the input $n_{ij} = \hat{Q}_{ij}(T_k)$ is upper bounded by $\sum_{ij} \hat{Q}_{ij}(T_k)$. We obtain that the optimal value $C_{\max}^{k+1}$ of the multicommodity flow problem on the input $n_{ij} = Q_{ij}(T_{k+1}) = A_{ij}(T_k, T_{k+1}) + \hat{Q}_{ij}(T_k)$ satisfies

$$C_{\max}^{k+1} \le r(T_{k+1} - T_k) + w + |V|^2|E|\sqrt{C_{\max}^{k}} = r(C_{\max}^{k} + |V|\sqrt{C_{\max}^{k}}) + w + |V|^2|E|\sqrt{C_{\max}^{k}}.$$

This proves (16). From (16) we obtain

$$\limsup_{k\to\infty} C_{\max}^{k} \le r \limsup_{k\to\infty} C_{\max}^{k} + (r|V| + |V|^2|E|) \limsup_{k\to\infty} \sqrt{C_{\max}^{k}} + w$$

$$\le r \limsup_{k\to\infty} C_{\max}^{k} + (r + |V|^2|E| + w)\sqrt{\limsup_{k\to\infty} C_{\max}^{k}}$$

or

$$\limsup_{k\to\infty} C_{\max}^{k} \le \frac{(r + |V|^2|E| + w)^2}{(1-r)^2}. \tag{18}$$

We finally argue that, for any $t = 0, 1, 2, \ldots$, the total number of packets in the network at time t, for large t, is at most $2|E|C_{\max}^{k_t}$, where k_t satisfies $T_{k_t} \le t < T_{k_t+1}$. We split all the packets in the network at time t into two groups: those that arrived before time T_{k_t} and those that arrived after time T_{k_t}. The first group has a size at most $\sum_{ij} Q_{ij}(T_{k_t})$, which is at most $|E|C_{\max}^{k_t}$ since $C_{\max}^{k_t}$ is a feasible solution to the multicommodity problem on the input $n_{ij} = Q_{ij}(T_{k_t})$. The second group has a size at most $\sum_{ij} Q_{ij}(T_{k_t+1})$ since none of these packets are processed before time T_{k_t+1}. Therefore, the second group has a size at most $|E|C_{\max}^{k_t+1}$. We apply (18) and conclude the proof of the theorem.

We now prove that if $r > 1$, then no stable routing policy exists. For that we need to exhibit a certain adversarial arrival pattern with arrival rate r for which stability cannot be achieved. Select $\delta > 0$, a small positive value, and construct a set of rates r_{ij} such that the optimal value r_0 of the multicommodity flow problem on the input $n_{ij} = r_{ij}$ satisfies $r_0 = r/(1+\delta)$. This can be achieved as follows: select values r'_{ij} very small arbitrarily, and increase them proportionally by t, $r_{ij} = r'_{ij}t$ until the objective value becomes r_0. We now construct an arrival pattern. For each pair (i,j), inject a type (i,j) packet every $1/r_{ij}$ time units. Select an integer $w > 0$ so that $r_{ij}w > 1/\delta$ for every r_{ij}. Then, for every time t,

$$A_{ij}(t, t+w) \le \left\lceil \frac{w}{1/r_{ij}} \right\rceil \le r_{ij}w + 1 = r_{ij}w + \frac{1}{r_{ij}w} r_{ij}w \le r_{ij}(1+\delta)w.$$

Since the optimal value of the multicommodity flow problem on the input $r_{ij}(1+\delta)$ is $r_0(1+\delta) = r$, the network is of the (r, w, FMF) type. In the proof of Corollary 1, we show that, for this arrival pattern, if the objective value $r_0 = r/(1+\delta) > 1$, then no stable policy can exist. By choosing δ sufficiently small, we obtain the result. □

The Discrete Review Algorithm is one way of turning a schedule for a static packet routing problem into a schedule for a dynamic packet routing problem. There are schedules other than the PRSA which achieve a certain degree of closeness to the optimal makespan time. For example, a routing schedule was constructed in [15] with makespan time $cC_{\max}$, where $C_{\max}$ is the optimal maximal congestion (solution to the multicommodity flow problem), and c is some (large) constant independent of the data of the problem. Note that this schedule, if implemented in the Discrete Review manner, does not necessarily lead to a stable policy if $r > 1/c$. The asymptotic optimality of the routing schedule (which is achieved by the PRSA) is essential for stability.

The result above can be used to decide what the necessary and sufficient conditions are for stability in adversarial queueing networks when the arrival rate for each pair of origin-destination is bounded by a constant.

COROLLARY 1. *Given an adversarial queueing network (V, E), suppose that there exist constants r_{ij}, $i, j \in V$, and a positive integer $w > 0$ such that the total number of packets injected during any interval $[t_1, t_2)$ with origin-destination pair (i, j) is not bigger than $r_{ij}(t_2 - t_1) + w$. If the multicommodity flow problem on the input r_{ij}, $i, j \in V$, has an optimal solution satisfying $C_{\max} < 1$, then there exists a stable packet routing schedule. If the optimal solution satisfies $C_{\max} > 1$, then a stable schedule cannot exist.*

Proof. Suppose $C_{\max} < 1$ for the multicommodity flow problem on the input $n_{ij} = r_{ij}$. Select a positive integer

$$W > \frac{2|V|^2 w}{1 - C_{\max}}.$$

By assumption, for any time t,

$$A_{ij}(t, t+W) \le r_{ij}W + w.$$

Also, by assumption, the optimal value of the multicommodity flow problem on the input $n_{ij} = r_{ij}W$ is at most $C_{\max}W$. The optimal value of the multicommodity flow problem on the input $n_{ij} = w$ is trivially at most $|V|^2 w$. Therefore, the optimal value of the multicommodity flow problem on the input $n_{ij} = r_{ij}W + w$ is at most

$$C_{\max}W + |V|^2 w = C_{\max}W + \frac{|V|^2 w}{W} W \le C_{\max}W + \frac{1 - C_{\max}}{2} W = \frac{1 + C_{\max}}{2} W.$$

We set $r = (1 + C_{\max})/2 < 1$. Then our queueing network is of the type (r, W, FMF). We apply Theorem 4 and complete the proof of the first part.

Suppose now that the optimal solution to the multicommodity flow problem on the input $n_{ij} = r_{ij}$ satisfies $C_{\max} > 1$. Let an adversary inject a type (i, j) packet every $1/r_{ij}$ time unit. Then, for every type (i, j),

$$A_{ij}(t_1, t_2) \le \left\lceil \frac{t_2 - t_1}{1/r_{ij}} \right\rceil \le r_{ij}(t_2 - t_1) + 1 \le r_{ij}(t_2 - t_1) + w.$$

So this arrival pattern satisfies the conditions of the theorem. Suppose, for the purposes of contradiction, that there exists a stable routing policy. For every type (i, j) and every edge $(k, l) \in E$, let $D_{kl}^{ij}(t_1, t_2)$ denote the number of type (i, j) packets that crossed the edge (k, l) during the time interval $[t_1, t_2)$, when this stable policy is implemented. Let also $Q_k^{ij}(t)$ denote the number of type (i, j) packets that are queued at the node k at time t. Then

$$Q_i^{ij}(t) = Q_i^{ij}(0) + A_{ij}(0, t) - \sum_{k:(i,k)\in E} D_{ik}^{ij}(0, t). \tag{19}$$

Also, for each $k \neq i, j$,

$$Q_k^{ij}(t) = \sum_{l:(l,k)\in E} D_{lk}^{ij}(0, t) - \sum_{l:(k,l)\in E} D_{kl}^{ij}(0, t). \tag{20}$$

By assumption, $Q_j^{ij}(t) = 0$ and $D_{jl}^{ij}(0, t) = 0$ for all l such that $(j, l) \in E$. Since the policy implemented is stable, then there exists $B > 0$ such that, for all k and all times t, $Q_k^{ij}(t) \le B$. In particular,

$$Q_k^{ij}(t)/t \to 0 \tag{21}$$

when $t \to \infty$. Note that, for each $(k, l) \in E$,

$$\sum_{i,j} D_{kl}^{ij}(0, t)/t \le 1 \tag{22}$$

since each edge processes at most one packet at a time. Therefore, there exists a sequence $t_1 < t_2 < \cdots < t_s < \cdots$ along which all the limits

$$x_{kl}^{ij} \equiv \lim_{s\to\infty} \frac{D_{kl}^{ij}(0, t_s)}{t_s}$$

exist. Note also that

$$\lim_{t\to\infty} \frac{A_{ij}(0, t)}{t} = r_{ij}.$$

From (19)–(21), it follows that x_{kl}^{ij}, $i, j \in V$, $(k, l) \in E$ is a feasible solution to the multicommodity flow problem on the input $n_{ij} = r_{ij}$. However, from (22) it follows that, for each edge $(k, l) \in E$,

$$\sum_{i,j} x_{kl}^{ij} \le 1.$$

In particular, the maximal congestion $C_{\max}$ corresponding to this feasible solution satisfies $C_{\max} \leq 1$. This contradicts the assumption. □

It is not clear whether a stable packet routing schedule exists for the critical case $C_{\max} = 1$, analogous to the NTO policy proposed in section 3. The existence of such a policy might depend on the integrality of a feasible solution to the multicommodity flow problem (6)–(12).

Note that the bound given by Theorem 4 is weaker than the bound

$$O\left(\frac{|V|^{5/2}|E|^{5/2}w}{1-r}\right) \tag{23}$$

obtained in [1] for networks of type (r, w, IMF). Also, the schedule constructed in [1] is distributed, whereas our schedule needs information about the entire network at times T_k. This raises the question of whether the schedule in [1] is applicable for the network of type (r, w, FMF). This turns out to be possible but at the cost of a somewhat inferior performance. Consider[1] an optimal fractional solution (maximal congestion) $C_{\max}$ to the multicommodity problem on the input $n_{ij} = A_{ij}(t, t+w)$. Using Raghavan's and Thomson's randomized algorithm (see [13]), one can construct an integral solution to the multicommodity flow problem with expected maximal congestion satisfying $C^{\text{int}}_{\max} \leq C_{\max} + C_{\max} \cdot O(\sqrt{\frac{\log|E|}{C_{\max}}})$. Suppose now that

$$w > O\left(\frac{\log|E|r}{(1-r)^2}\right). \tag{24}$$

Then $C_{\max} \leq rw$ implies $C^{\text{int}}_{\max} \leq rw + \sqrt{rw}O(\sqrt{\log|E|}) < w$. Thus, if (24) holds, then the network is of the type (r', w, IMF) for $r' \equiv C^{\text{int}}_{\max}/w$. Unfortunately, the bound (24) combined with (23) implies the bound

$$O\left(\frac{|V|^{5/2}|E|^{5/2}\log|E|r}{(1-r)^2(1-r')}\right),$$

which is at best (using $r \leq r'$) $O(\frac{1}{(1-r)^3})$ when $r \approx 1$. This is inferior to our bound $O(\frac{1}{(1-r)^2})$ in Theorem 4 when $r \approx 1$. (Recall that our bound does not require (24).)

Observe that we used the second part of Theorem 3 in the construction of the Discrete Review Algorithm. We could also use the first part (routing entirely the packets present in the system at time T_k) at the price of somewhat larger upper bounds, although this scheme has the following advantage. In this form of implementation, the path of each packet is predetermined at time T_k for packets present in their origination node at time T_k. Thus the path information can be recorded in the header of the packet, and no path recalculation is needed while the packet is on its route.

5. Conclusion. We have provided in this paper a simple classification of undirected graphs which are universally stable: a connected undirected graph is universally stable if and only if it has at most two edges. We have also proved that a simple distributed policy, NTO, achieves stability in all the graphs under the critical arrival rate $r = 1$. A multicommodity flow-type load condition was formulated for adaptive adversarial queueing networks, and a stable policy was constructed whenever this load condition was met.

[1]The author wishes to thank Ashish Goel for pointing out this argument.

A number of interesting questions remain outstanding. It is not clear which graphs are universally stable for a given value of $r < 1$. Given that a network could be unstable for an arbitrarily small r (see [6]), such a classification could be quite nontrivial. Deciding the stability of specific policies is not well understood in general and might become an impossible problem in light of the undecidability results in [10].

Acknowledgments. The author wishes to thank Ashish Goel, Matthew Andrews, and several anonymous referees for many fruitful suggestions and corrections.

REFERENCES

[1] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosen, *Adaptive packet routing for bursty adversarial traffic*, J. Comput. System Sci., 60 (2000), pp. 482–509.

[2] M. Andrews, *Instability of FIFO in session-oriented networks*, in Proceedings of the 11th ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2000, pp. 440–447.

[3] M. Andrews, B. Awerbuch, A. Fernandez, J. Kleinberg, T. Leighton, and Z. Liu, *Universal stability results for greedy contention-resolution protocols*, in Proceedings of the 27th IEEE Conference on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1996, pp. 380–389.

[4] M. Andrews and L. Zhang, *The effects of temporary sessions on network performance*, in Proceedings of the 11th ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2000, pp. 448–457.

[5] D. Bertsimas and D. Gamarnik, *Asymptotically optimal algorithm for job shop scheduling and packet routing*, J. Algorithms, 33 (1999), pp. 296–318.

[6] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. Williamson, *Adversarial queueing theory*, J. ACM, 48 (2001), pp. 13–38.

[7] M. Bramson, *Instability of FIFO queueing networks*, Ann. Appl. Probab., 2 (1994), pp. 414–431.

[8] R. Cruz, *A calculus for network delay, part* II: *Network analysis*, IEEE Trans. Inform. Theory, 37 (1991), pp. 132–141.

[9] D. Gamarnik, *Using fluid models to prove stability of adversarial queueing networks*, IEEE Trans. Automat. Control, 4 (2000), pp. 741–747.

[10] D. Gamarnik, *On deciding stability of constrained homogeneous random walks and queueing systems*, Math. Oper. Res., 27 (2002), pp. 272–293.

[11] A. Goel, *Stability of networks and protocols in the adversarial queueing model for packet routing*, in Proceedings of the 10th ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 911–912.

[12] J. Kleinberg, *personal communication*, Cornell University, Ithaca, NY, 1998.

[13] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[14] N. Robertson and P.D. Seymour, *Recent Results on Graph Minors*, Cambridge University Press, Cambridge, UK, 1985.

[15] A. Srinivasan and C.P. Teo, *A constant-factor approximation algorithm for packet routing, and balancing local vs. global criteria*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 636–643.

[16] L. Tassiulas and L. Georgiadis, *Any work-conserving policy stabilizes the ring with spatial reuse*, IEEE/ACM Trans. Networking, 4 (1996), pp. 205–208.

[17] P. Tsaparas, *Stability in Adversarial Queueing Theory*, M.Sc. thesis, University of Toronto, Toronto, Canada, 1997.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 386–407

MANY TO ONE EMBEDDINGS FROM GRIDS INTO CYLINDERS, TORI, AND HYPERCUBES*

JOHN ELLIS†, STIRLING CHOW†, AND DENNIS MANKE†

Abstract. We describe novel methods for embedding 2-dimensional grid graphs into cylinders (one way wrap-around grids), tori, and hypercubes, where the guest grid G is larger than the host graph H, implying a many to one embedding.

We call $\lceil \mid G \mid / \mid H \mid \rceil$ the optimal load, denoted l. We consider optimal embeddings with respect to dilation (the stretching of guest edges) and load; i.e., edges are mapped to edges or onto one node, and the number of grid nodes mapped onto any hypercube node is not greater than l.

We show, by construction, that, for loads of at least 4, optimal embeddings into the hypercube always exist subject only to modest restrictions on the relative dimensions of guest and host. If the problem instances are grouped by grid height, the restrictions imply that only some finite number of instances in each group may not be solvable by the given methods.

The essence of the method is a mapping from grids into cylinders of height at least one half of but not greater than the grid height, and so it can also be used to construct embeddings into cylinders and tori.

Previous work has gone so far as to show that if the optimal load l is a power of 2, then dilation 1, load $l + 1$ embeddings into the hypercube can be constructed. Optimal results for loads 2 and 3 are not known.

Key words. graph embedding, many to one, grid, cylinder, torus, hypercube

AMS subject classification. 05C10

PII. S0097539700377657

1. Introduction.

The study of embeddings from one family of graphs to another has received substantial attention over many years. Besides the inherent mathematical interest of these problems, motivation for the study comes from the many concerns of computer science that can be effectively modelled by this graph theoretical abstraction. In this paper, we describe some results on many to one embeddings from 2-dimensional grids into smaller cylinders, tori, and hypercubes. Plausible applications for our results include the simulation of large grid-like networks of processors by smaller networks in the form of cylinders, tori, or hypercubes and the efficient implementation of a parallel algorithm conceived of as running on a grid of unlimited size but executed on some fixed sized network in those same families. In these applications the embedding parameters *load*, *dilation*, and *congestion*, defined in the next section, strongly influence the effectiveness of the simulation.

Several papers have discussed grid to grid and grid to hypercube embeddings. [6, 7, 9, 15] describe one to one embeddings from 2-dimensional grids into other 2-dimensional grids of smaller aspect ratio, with dilation 2. Likewise, [12] considers efficient embeddings from grids into grids.

One to one embeddings from 2-dimensional grids into hypercubes are considered in [13], and those from 3-dimensional grids into hypercubes are considered in [3] and [11].

*Received by the editors September 1, 2000; accepted for publication (in revised form) October 7, 2002; published electronically January 28, 2003. This work was supported by the Natural Sciences and Engineering Research Council of Canada.

http://www.siam.org/journals/sicomp/32-2/37765.html

†Department of Computer Science, University of Victoria, P.O. Box 3055, Victoria, British Columbia, V8W 3P6, Canada (jellis@cs.uvic.ca, schow@cs.uvic.ca).

It is well known (see, for example, [8] and Lemma 3.1) that not all 2-dimensional grids are subgraphs of the minimum sized hypercube sufficient to contain the grid. In [2], it is shown that all 2-dimensional grids can be one to one embedded into the smallest possible hypercube with dilation two. This result can also be obtained by an entirely different route, namely, by using the grid to grid embeddings just cited to reduce the width of the grid down to the next power of two; i.e., embed the guest grid into a grid that is a subgraph of the hypercube. This is the approach taken in [1].

What happens when the guest grid is arbitrarily larger than the host graph, i.e., one considers many to one embeddings, is by no means completely understood. Do dilation one embeddings with an optimal load always exist, at least into the hypercube if not for the simpler families?

Many to one embeddings from 2-dimensional grids into smaller 2-dimensional grids are considered in [14]. There it is shown that dilation one with optimal load plus one is always obtainable as long as the host is smaller than the guest in both dimensions. In [10], many to one embeddings from both 2- and 3-dimensional grids into hypercubes are considered. For 2-dimensional grids, the main result (Theorem 1) states the following (and we paraphrase): *If the optimal load is* 2^k *for some integer* k*, then an embedding exists with dilation one and load* $2^k + 1$.

Our main result shows that dilation one optimal load solutions can always be constructed for embeddings from the 2-dimensional grid into the hypercube for all loads greater than three, with only a modest restriction on the relative dimensions of the guest and host. The restriction may be interpreted as saying that, in each group of problem instances of a given grid height, only a finite number might not be solved by our methods.

The technique is based on a mapping into the torus, so similar results also apply to the torus, under some restrictions on the relation between grid and torus dimensions.

For cylinders, the absence of horizontal wrap-around edges seems to prevent the technique from achieving optimal load in those instances where the host is only just big enough to contain the guest.

We leave open the conjecture that instances of the grid to hypercube embedding problem always have optimal load and dilation 1 solutions for all loads greater than or equal to 2 and irrespective of the relative dimensions of the guest and host.

2. Definitions. A *graph embedding* is a mapping from the nodes of the guest graph to the nodes of the host graph, together with a mapping from guest edges to paths in the host. In general, the node mapping can be one to one or many to one, but here we are concerned only with many to one embeddings. The *dilation* of a guest edge is the length of the path which is the image of that edge in the host. Because, in a many to one mapping, nodes incident to an edge could be mapped to the same host node, it is possible that the dilation of an edge is zero. The dilation of an entire embedding is the largest dilation over all edges in the guest.

Here we consider only instances where the number of nodes in the guest graph, denoted $\mid G \mid$, is greater than the number of nodes in the host, $\mid H \mid$. For any many to one mapping, the maximum over all host nodes of the number of guest nodes mapped to the host node is called the *load* of the mapping. We call the ratio $\lceil \mid G \mid / \mid H \mid \rceil$, denoted l, the *optimal load*. Throughout the paper, we assume that $l \geq 2$.

The *congestion* of an embedding is the maximum over all host edges of the number of guest edges mapped to a path that includes the host edge.

We are interested only in what we call *optimal* embeddings, which are those with dilation 1 and for which no more than l guest nodes are mapped onto any host node.

Restricting the dilation to 1 means that we consider only embeddings that map edges to edges or map both ends of a guest edge onto the same host node. Hence we have no need in this presentation to explicitly define the guest edge to host path mapping.

The family of guest graphs which we consider consists of the 2-dimensional grids. A 2-dimensional grid of height h and width w is the graph comprising the node set $\{(x, y) \mid 0 \leq x < h, 0 \leq y < w\}$ and the edge set $\{\{(u, v), (x, y)\} \mid \mid u - x \mid + \mid v - y \mid = 1\}$. We will refer to these guest graphs as (h, w)-*grids*.

The families of host graphs considered are as follows:

1. Cylinders, which we picture as (h, w)-grids with wrap-around edges in the vertical dimension, are the graphs comprising the node set $\{(x, y) \mid 0 \leq x < h, 0 \leq y < w\}$ and the edge set $\{\{(u, v), (x, y)\} \mid \mid u - x \mid + \mid v - y \mid = 1\} \bigcup \{(0, v), (h - 1, v)\}$. We will refer to these host graphs as (h, w)-*cylinders*.

2. Tori, which we picture as (h, w)-grids with wrap-around edges in both dimensions, are the graphs comprising the node set $\{(x, y) \mid 0 \leq x < h, 0 \leq y < w\}$ and the edge set $\{\{(u, v), (x, y)\} \mid \mid u - x \mid + \mid v - y \mid = 1\} \bigcup \{(0, v), (h - 1, v)\} \bigcup \{(u, 0), (u, w - 1)\}$. We will refer to these host graphs as (h, w)-*tori*.

3. Hypercubes of dimension d, denoted Q_d, are the graphs with vertex set $\{n \mid 0 \leq n \leq 2^n - 1\}$ and edge set $\{\{x, y\} \mid$ the binary representations of x and y differ in exactly one bit position$\}$.

3. An example. Figure 3.1 defines our notation for naming the nodes of the guest and host graphs. We begin with a small example by way of introducing both our notation and our method. In this example, the host is a cylinder.

It is well known (see, for example, [8]) that not all 2-dimensional grids G are subgraphs of H, where H is the smallest hypercube such that $|G| \leq |H|$. The distinction between those that are and are not is defined in the following lemma.

LEMMA 3.1. *The (h, w)-grid is a subgraph of Q_d iff $\lceil \log_2 h \rceil + \lceil \log_2 w \rceil \leq d$.*

So, for example, the $(5, 6)$-grid is not a subgraph of Q_5. We also use another well-known property of the hypercube Q_d; again, see [8].

LEMMA 3.2. *If $p + q \leq d$, then the $(2^p, 2^q)$-torus is a subgraph of Q_d.*

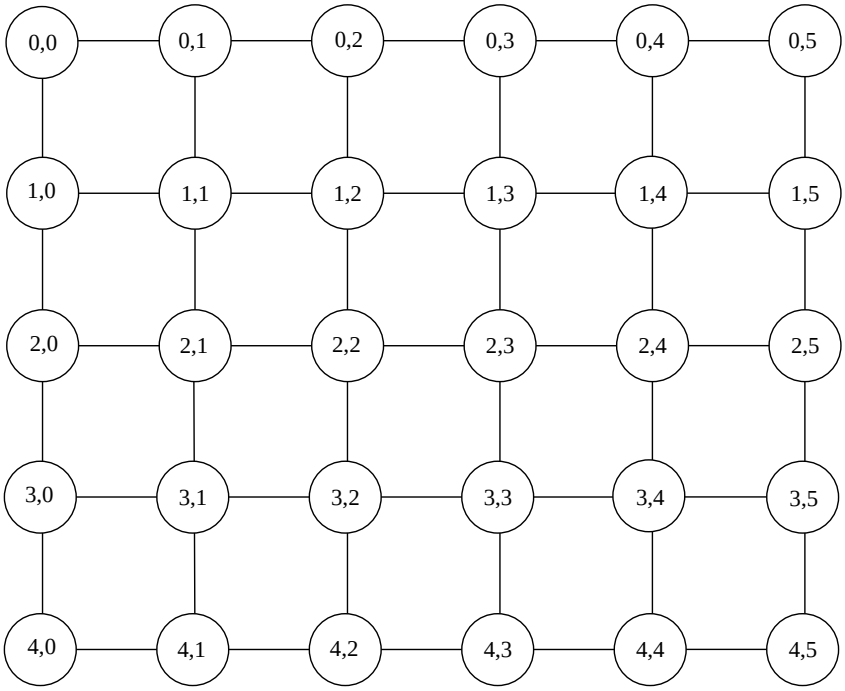
Consequently, the $(4, 8)$-cylinder is a subgraph of Q_5. Hence there is no dilation 1 load 1 embedding from the $(5, 6)$-grid into the $(4, 8)$-cylinder because such an embedding would imply the existence of a dilation 1 embedding from the (5,6)-grid into Q_5, contradicting Lemma 3.1.

However, there is a load 2 dilation 1 embedding into the $(4, 4)$-cylinder. A solution, illustrated in Figure 3.2, demonstrates the way we will illustrate all such embeddings. If the guest is an (h, w)-grid, the diagram shows an $h \times w$ array of integers. Let the top leftmost element in the array shown in the figure be indexed (0,0). Then any element (i, j) in the array represents a node (i, j) in the guest grid, as illustrated in Figure 3.1. The value of array element (i, j) defines which host row is to be the image of grid node (i, j). The host column number is defined by the numbers beneath the bottom row, where adjacent host columns are distinguished by the presence or absence of shading. For example, Figure 3.2 tells us that node (4,3) of the guest is mapped to row 2, column 1 of the host.

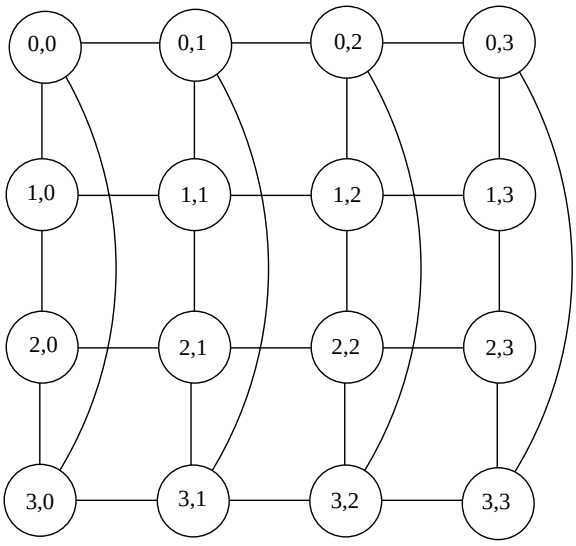
Remembering that there exist wrap-around edges between rows 0 and 3 in the host cylinder, one can verify that the diagram defines a legitimate load 2 embedding in which edges are mapped to edges by noticing the following:

1. Each host node appears no more than twice.

2. Elements in the array that are vertically or horizontally adjacent, i.e., representing adjacent guest nodes, are designated either adjacent row numbers in the same



a) A (5,6)-grid



b) A (4,4)-cylinder

FIG. 3.1. *Node-naming conventions.*

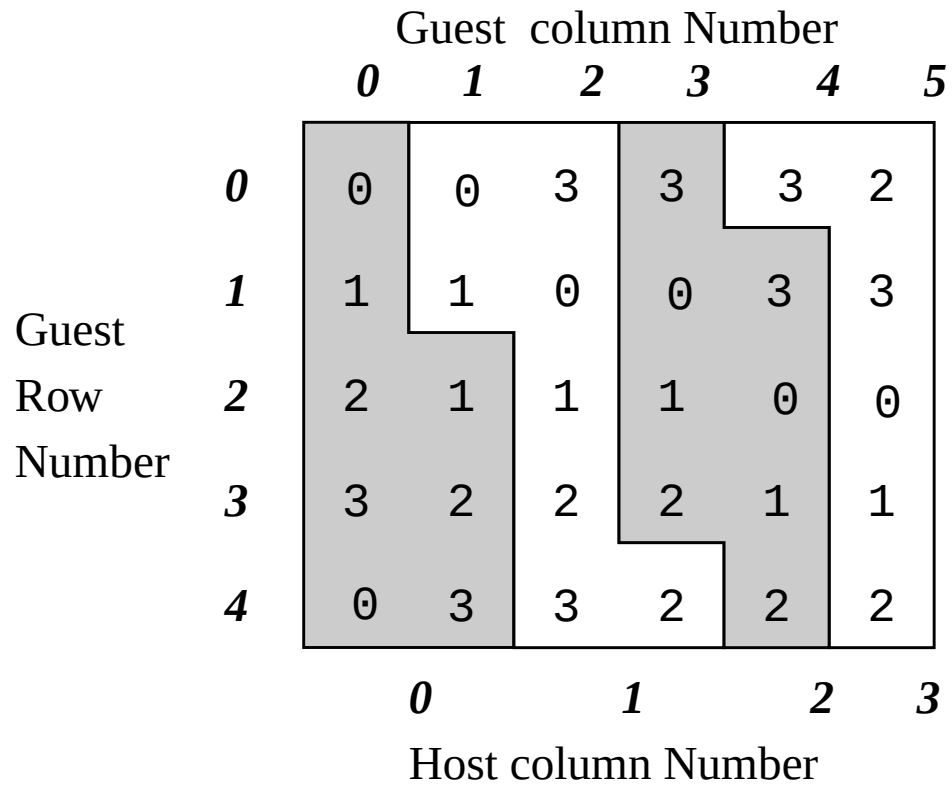


FIG. 3.2. *Embedding a* $(5,6)$*-grid into a* $(4,4)$*-cylinder with load* 2.

host column or identical row numbers in adjacent host columns.

Our way of visualizing embeddings is an alternative to that often used in previous work, where one draws the host and indicates for each host node which guest nodes are mapped onto it. It is interesting to note that the solutions suggested by these different views are quite distinct and that the pattern underlying one type of solution may not be at all apparent in the alternative visualization.

4. The partial mapping.

4.1. Informal description of the partial mapping. Our embedding is based on a single, simple guest node to host node mapping. This mapping does not immediately give us the final embedding because it is only a partial mapping from the guest (h,w)-grid. This partial mapping does not define an image for guest nodes in the bottom left and top right corners in the diagrams. The final total mapping is constructed from the partial mapping by methods to be described in later sections. In this section, we describe and define the partial mapping and investigate some of its properties.

The partial mappings are illustrated in the same way as in the earlier example, Figure 3.2. Consider Figure 4.1, which defines a load 2 dilation 1 mapping from the nodes of a subgraph of a $(13,20)$-grid to the nodes of an (8,13)-cylinder.

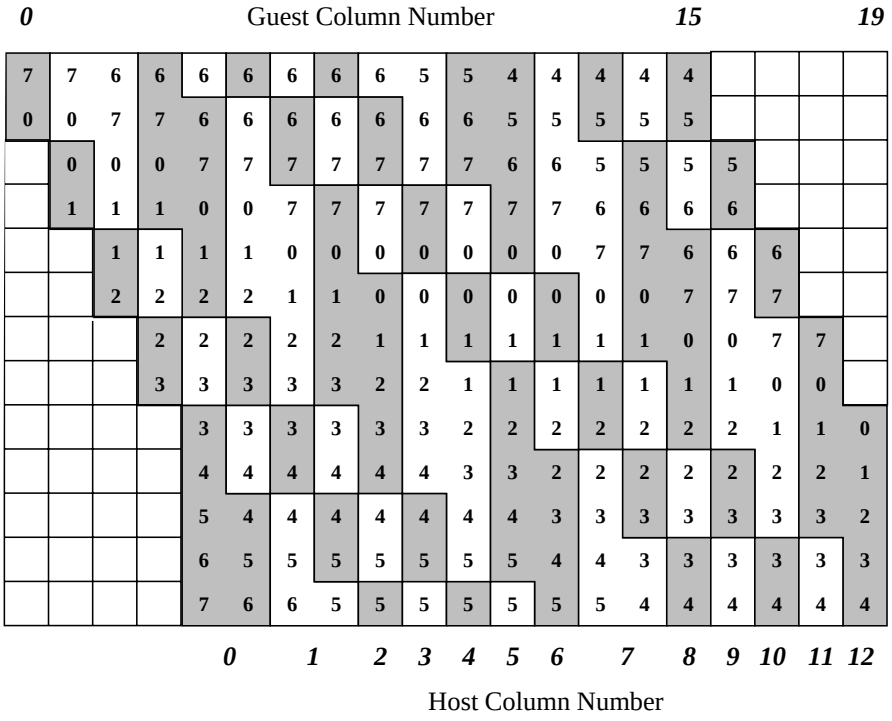


FIG. 4.1. *A height* 13 *load* 2 *partial mapping.*

4.2. Formal definition of the partial mapping. The partial mapping maps nodes from a grid of height h_g onto the nodes of a cylinder of height h_h, where $h_g/2 < h_h < h_g$, with load l. To completely define the mapping, we need a fourth parameter, which we call a *profile*. Let $\mathbf{p} = p_1 p_2 \cdots p_k$ denote a string of k integers.

DEFINITION 4.1. *The string* $\boldsymbol{p}$ *is a* (k, n)*-profile if* $\sum_{i=1}^{k} p_i = n$*, and, for all* i*,* $p_i \geq 2$*.*

Informally, the profile associated with a given mapping is the sequence of "step" heights on the left or right of a host column in a mapping diagram. We will refer to these step sequences as left and right column profiles when the context makes the meaning clear. To map a height h_g guest into a height h_h host, we need an $(h_g - h_h, h_g)$-profile. Of course, for any h_g, h_h pair, there are many profiles satisfying the definition. For example, the (5,13)-profile used to generate Figure 4.1 and which is the left profile of host column 0 is 2 2 2 2 5.

For each $(h_g, h_h, l, \mathbf{p})$ 4-tuple, we define a mapping, i.e., we define a function $F(h_g, h_h, l, \mathbf{p})$, whose value is the required node to node mapping.

For each column i of the host, $F(h_g, h_h, l, \mathbf{p})$ defines the set C_i of triples (r, c, v), where $0 \leq r < h_g$. The existence of the triple (r, c, v) in C_i indicates that node (r, c) of the guest is to be mapped to node (v, i) of the host; i.e., the contents of C_i define completely which guest nodes are mapped onto which row nodes of column i in the host. Note that i is unbounded, meaning that we can apply the mapping to any width host of the given height.

Before presenting a formal definition of the partial mapping, Definition 4.2 below, we give an informal description. Figure 4.1 can be used as an example. Let $k = h_g - h_h$ so that we use a profile $\mathbf{p} = p_1 p_2 \cdots p_k$.

1. Starting at the top of guest column 0, mark off p_1 rows in that column. In column 1, mark off the p_2 rows immediately following the marked rows in column 0. Continue in this fashion until the last p_k rows are marked in column $k-1$. By the definition of $\mathbf{p}$, h_g rows have been marked in the first k columns of the guest grid.

2. Consider the sequence of integers 0 through $h_h - 1$. Skipping the top row in each marked column segment, write the first $p_1 - 1$ numbers in the sequence in the remaining $p_1 - 1$ rows of column 0, the next $p_2 - 1$ numbers in the sequence in the remaining $p_2 - 1$ rows of column 1, and so on to the last $p_k - 1$ numbers in the sequence in the remaining $p_k - 1$ rows of column $k-1$. At this point, we have computed C_0^0 as defined in item 1 (i) of Definition 4.2.

3. Repeat $l-1$ times: run down the right-hand edge of the current pattern, and, for each of the most recent additions of the numbers 0 through $h_h - 1$, copy that number into the next location one column to the right and one row down from the current position. If moving down runs off the bottom of the pattern, place the number in the next vacant location in row 0. At this point, we have completed the computation of C_0, i.e., the mapping into the first host column, as defined in item 1 (ii) of Definition 4.2.

4. To create C_1 from C_0, i.e., host column 1 from host column 0, or, in general, C_{i+1} from C_i, all numbers are incremented by one, and the pattern is cyclically rotated downward by l rows. In the cyclic rotation, items that run off the bottom are returned to the top rows, but the column number is decreased by k every time the new row number passes through a multiple of h_g. This process is defined in item 2 of Definition 4.2.

Here is the algebraic definition, which is used to prove claims about the properties of the mapping and could be used as the basis for an algorithm to generate it.

DEFINITION 4.2 (the partial mapping). *The C_i corresponding to the 4-tuple $(h_g, h_h, l, \mathbf{p})$, where $\mathbf{p}$ is a (k,n)-profile, are defined recursively as follows:*

1. $C_0 = \bigcup_{j=0}^{l-1} C_0^j$, *where the following hold:*

(i) *Let $d_1 = 0$ and for all $i, 2 \le i \le k$, let $d_i = d_{i-1} + p_i - 1$. Then for all $i, 1 \le i \le k$; for all $z, d_i \le z < d_i + p_i - 1, (z+i, i-1, z) \in C_0^0$.*

(ii) *For all $j, 1 \le j \le l-1, (r,c,v) \in C_0^{j-1}$ implies $((r+1) \bmod h_g, c+1-\lfloor (r+1)/h_g \rfloor k, v) \in C_0^j$.*

2. *For all $i, 1 \le i$, $(r,c,v) \in C_{i-1}$ implies $((r+l) \bmod h_g, c+l-\lfloor (r+l)/h_g \rfloor k, (v+1) \bmod h_h) \in C_i$, where "mod" denotes the remainder after integer division.*

The reader will find that C_0, generated with $l = 2$, $h_g = 13$, and $h_h = 8$, and profile 2 2 2 2 5 yield host column 0 in Figure 4.1 and that C_1 defines host column 1 and so on. This definition can be used as the basis for an algorithm which computes the mapping using a number of arithmetic and other primitive operations linear in the size of the guest.

4.3. Properties of the partial mapping. The following properties of the mappings are evident from the diagrams and are straightforward to prove.

The mapping properties.

1. Every node in the host is the image of exactly l guest nodes; i.e., the load is optimal (taking the size of the domain of the mapping to be the numerator).

2. If two nodes are adjacent in the guest and both have images defined by the mapping, then their images are adjacent in the host; i.e., the dilation of the mapping is one.

3. Each edge in a host column appears no more than $2l-1$ times, and each edge in a host row appears no more than l times; i.e., the congestion of the mapping

is $\leq 2l-1$.

4. The left and right column profiles are rotated through l rows from one host column to the next.

The first statement follows directly from rule 1 (ii) in the definition of the mapping. The second can be demonstrated by an inductive argument based on noting that, within a column, adjacent array elements are given adjacent row numbers and that, across column boundaries, adjacent elements are given identical row numbers. The third statement follows from the fact that adjacent consecutive row numbers within a host column correspond to column edges and there are no more than $2l-1$ such adjacencies per edge. Host row edges correspond to adjacent numbers across host column boundaries. Along a boundary, each row number can appear no more than l times. The fourth statement follows directly from rule 2 in the definition of the mapping.

The partial mapping has two significant periodicity properties. For example, notice that in Figure 4.1 the left profile of host column 0 and the right profile of host column 12 (which is identical to the left profile of host column 13) are identical.

DEFINITION 4.3. *The* structural periodicity *of the mapping is the smallest integer, p_s, such that $\exists \delta_s$ for all $i(r,c,v) \in C_i$ implies $(r, c+\delta_s, (v+p_s) \bmod h_h) \in C_{i+p_s}$.*

Note also that the values in host column 13 (the next column in the mapping but not shown in the diagram) would not be identical to those in column 0. However, C_0 would be identical, in both profiles and content, to C_{104} ($104 = 8 \times 13$) were the mapping to be continued that far.

DEFINITION 4.4. *The* numerical periodicity *of the mapping is the smallest integer, p_n, such that $\exists \delta_n$ for all $i(r,c,v) \in C_i$ implies $(r, c+\delta_n, v) \in C_{i+p_n}$.*

The following lemmas show how these periodicities are related to the parameters that define the mapping. We refer to the greatest common divisor and to the lowest common multiple of two positive integers i and j as $gcd(i,j)$ and $lcm(i,j)$, respectively, and we use $i \mid j$ to mean i divides j.

LEMMA 4.5. *The structural periodicity of the pattern is given by $p_s = h_g/gcd(l,h_g)$, and the constant δ_s is given by $\delta_s = lh_h/gcd(l,h_g)$.*

Proof. Suppose the operation defined in rule 2 of the initial mapping definition is repeated $p_s = h_g/gcd(l,h_g)$ times on a triple (r,c,v), producing the triple (r',c',v').

Then the following hold:

1. $r' = (r+lp_s) \bmod h_g = (r+lh_g/gcd(l,h_g)) \bmod h_g = r$.

2. Let $k = h_g - h_h$. The column number c will be incremented, by l, p_s times and decremented, by k, the number of times the value of r, incremented by l at each iteration, passes through a multiple of h_g, namely, $\lfloor (r+lp_s)h_g \rfloor$ times. Noting that $\lfloor r/h_g + (l/h_g)(h_g/gcd(l,h_g)) \rfloor = l/gcd(l,h_g)$, we have

$$\begin{aligned} c' &= c + lh_g/gcd(l,h_g) - lk/gcd(l,h_g) \\ &= c + l(h_g-k)/gcd(l,h_g) \\ &= c + lh_h/gcd(l,h_g). \end{aligned}$$

3. $v' = (v+p_s) \bmod h_h$.

4. $h_g/gcd(l,h_g)$ is the smallest integer z with the required properties because z must satisfy $r = (r+lz) \bmod h_g$. Then it must be that $h_g \mid lz$, i.e., $l = ax$, $z = by$, where $xy = h_g$ for some integers a,b,x,y. Since z is minimal, it must be that $x = gcd(l,h_g)$. Hence $z = by = bh_g/x = bh_g/gcd(l,h_g) \geq h_g/gcd(l,h_g)$. ∎

LEMMA 4.6. *The numerical periodicity of the pattern is given by*

$$p_n = lcm(h_h, h_g/gcd(l,h_g)).$$

Proof. We require that $r' = r$, which implies that $p_n = ap_s$ for some integer a. Also, we must have $v' = v$, and hence $ap_s \bmod h_h = 0$. Hence $ap_s = lcm(h_h, p_s) = lcm(h_h, h_g/gcd(l, h_g))$. □

Note that both periodicities are independent of the profile **p** used to initiate the mapping.

For some of our constructions, we need a lower bound on what we will call the *column thickness* of a mapping.

DEFINITION 4.7. *The* column thickness *of a mapping is the minimum over all rows of the number of grid columns in the row mapped to the same host column.*

LEMMA 4.8. *The thickness of a partial mapping is* $\geq \lfloor l/2 \rfloor$.

Proof. The thickness of C_0 is $\geq \lfloor l/2 \rfloor$ because $C_0^0 \cup C_0^1$ includes every row at least once. Hence every two repetitions of rule 1 (ii) of the mapping definition adds at least one to the thickness. The later columns are formed by the rotation of C_0, which does not change its shape. □

5. Embedding into cylinders. So far we have described only partial mappings from guest grid nodes to the host cylinder nodes, but our goal is to define a total mapping for (h, w)-grids. Obviously, a partial mapping can yield a total mapping for any (h, w)-grid whose nodes fall within the domain of the partial mapping. By a *partially mapped* guest column, we mean one for which one or more but not all column nodes have an image defined by the mapping. For example, in Figure 4.1, we can obtain a load 2, dilation 1 embedding of the $(13, 12)$-grid into an $(8, 13)$-cylinder by discarding the partially mapped columns.

In this section, we improve on that by showing that we can "square up" at least one end of the mapping diagram. This squaring up method works for one particular family of profiles. Let $k = h_g - h_h$. The (k, h_g)-profile we need is a sequence of $k - 1$ twos followed by $h_g - 2(k - 1)$. Let us call these *cylindrical* profiles. Figure 4.1 was initiated by such a profile.

A guest node will be said to be *uncovered* if it is not in the domain of the partial mapping. In the mapping diagrams, the uncovered nodes that we consider now are those making up the bottom left corner. We will refer to this set of uncovered nodes as the *corner*.

LEMMA 5.1. *There are* $(k-1)h_h$ *nodes in the corner of a partial mapping initiated by a cylindrical profile.*

Proof. We deduce from the definition of the partial mapping, rule 1 (i), that there are $k - 1$ partially mapped columns, that the leftmost such column is of height $h_g - 2$, and that the rightmost column is of height $h_g - 2k + 2$, decreasing by 2 at each column. A simple summation yields the following: the number of uncovered nodes $= (k - 1)(h_g - k) = (k - 1)h_h$. □

The remainder of the paper is presented in terms of mapping diagrams. We do not attempt to specify the mappings algebraically. The following process creates a corner mapping diagram which maintains the necessary mapping properties for any load l.

1. Take $(k - 1)$ height h_h columns, numbered 0 through $k - 2$. Assign the first l columns to host column 0, the next l to host column 1, and so on until all are given some designation. The columns will now be divided into $\lceil (k - 1)/l \rceil$ groups, with l columns per group, except perhaps for the last group, which will contain $(k-1) \bmod l$ columns. Number the rows of column 0 from 0 downward to $h_h - 1$ (or any rotation thereof). See Figure 5.1 (a), where an appropriate rotation is chosen to match Figure 4.1.

2. For columns 1 through $k-2$, the following hold.

(i) If the guest node and its left neighbor are assigned to the same host column, assign a row number one less (cyclically) than its left neighbor.

(ii) If the column and its left neighbor are assigned to different (adjacent) host columns, assign a row number identical to that of its left neighbor.

See Figure 5.1 (b).

3. Repeat the following *shift* process $k-2$ times: Consider a 45 degree diagonal drawn from above the top leftmost element, down to the right edge of the current profile. Shift everything above this diagonal one place up and one place to the left. See Figure 5.1 (c) through (e).

We observe the following.

1. The diagram covers $h_h(k-1)$ nodes, i.e., the number originally uncovered, and distributes them with load $\leq l$ across $\lceil (k-1)/l \rceil$ host columns.

2. After steps 1 and 2, mapping properties 2 and 3 hold.

3. Mapping properties 2 and 3 are preserved at each step of the shifting process. This can be seen by noting the following.

(i) Any diagonal shift involving elements originally related in the way defined by steps 1 and 2 preserves the properties. Consider two horizontally adjacent elements, *left* and *right*, on either side of the diagonal which defined the shift. Shifting puts *right* above *left*, and the element which was below and to the right of *right* becomes the right neighbor of *left*. In each of the only two possible cases (i.e., *left* and *right* are assigned either to the same or to adjacent host columns) the row numbering defined in steps 1 and 2 ensures that mapping properties are preserved.

(ii) All diagonal shifts are on elements related as defined in steps 1 and 2, as the diagonal moves up through previously undisturbed elements.

4. Height $h_g - 2$ is achieved.

5. The right profile of the result is a cylindrical profile.

Finally, we note that once a corner has been built in this way, we can continue the mapping in the usual way. It is necessary only to note whether those uncovered elements adjacent to elements covered by the corner building process should be assigned to the same host column or to the next one.

Case 1. $l \mid (k-1)$, i.e., the number of host columns used in the corner is a multiple of the load. In this case, the mapping that follows the corner begins with a new host column. Then the mapping definition 1 (i) is initiated not by a profile but by the numbers already defined by the corner. Then definition 1 (ii) is applied without change.

Case 2. Not $(l \mid (k-1))$, i.e., the available repetitions of the current host column, as determined by the load, have not been completely used. Again, rule 1 (i) is modified so that C_0^i, for some i, is defined not by a profile but by the numbers defined by the corner construction. C_0^{i+1} through C_0^{l-1} are defined as usual by rule 1 (ii).

Figure 5.1 shows the construction of a corner for the problem instance $h_h = 8$, $h_g = 13$, and load 2. Here we need four columns to fill the corner, i.e., two host columns, each appearing twice.

Figure 5.2 shows a similar example—the construction of a corner for the same problem instance but with load 3. Now we need two host columns, one appearing three times and the other once. As the mapping continues, it will first use two copies of this latter host column.

In general, we know how to square up the other end of the mapping only in certain

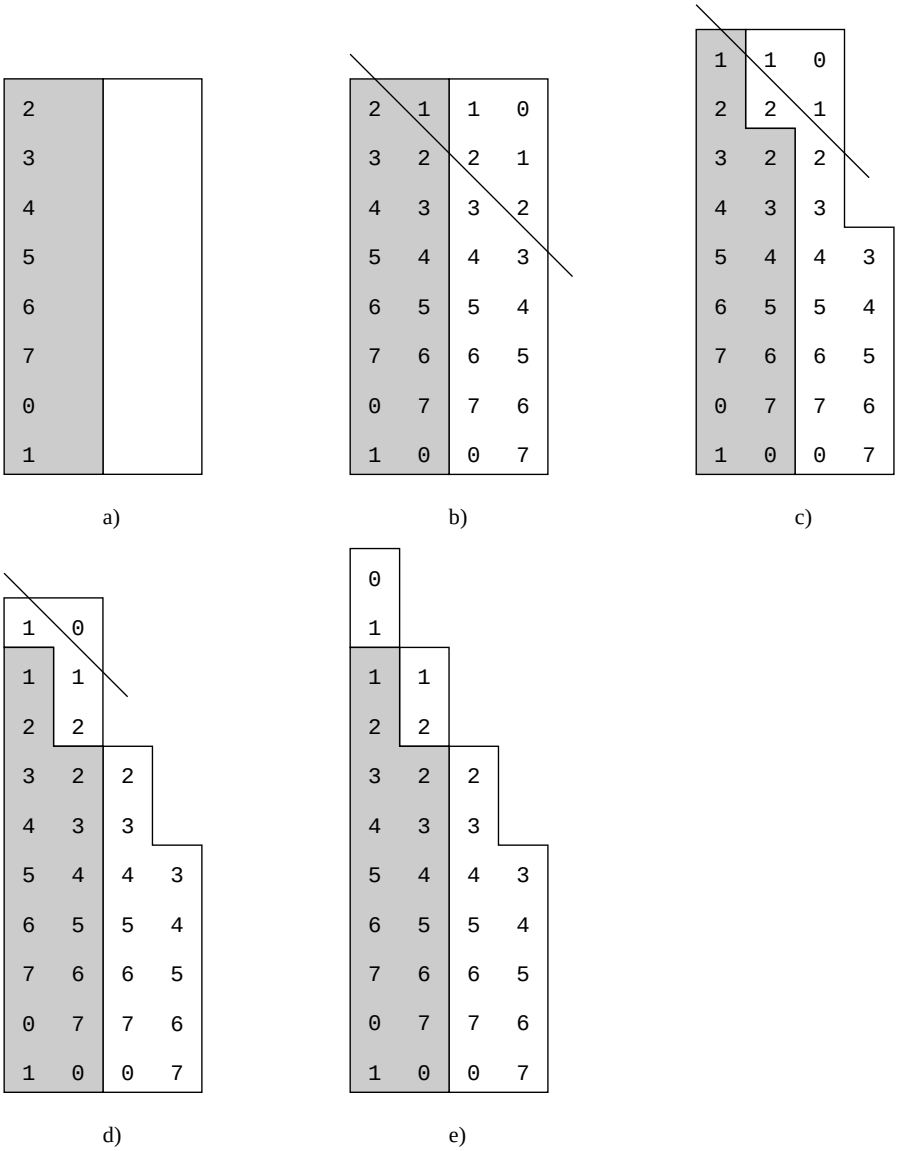


FIG. 5.1. *Creating a corner:* $h_g = 13$, $h_h = 8$, *load* 2.

special cases. We can, however, put an upper bound on how many extra host columns we may have to use to make up for the uncovered nodes in the top right corner of the mapping. We need to consider the *skew* of the mapping.

DEFINITION 5.2. *Let the mapping be defined for host columns* 0 *through* i. *The* skew *of a mapping is the number of guest columns that are partially mapped and that contain some element mapped to host column* i.

LEMMA 5.3. *The skew of any partial mapping is* $\leq h_g - h_h$.

Proof. By definition of the mapping, rule 1 (i), there are $h_g - h_h - 1$ partially mapped guest columns containing some element mapped to host column 0. As the mapping progresses across host columns, the profile can be split by the wrap-

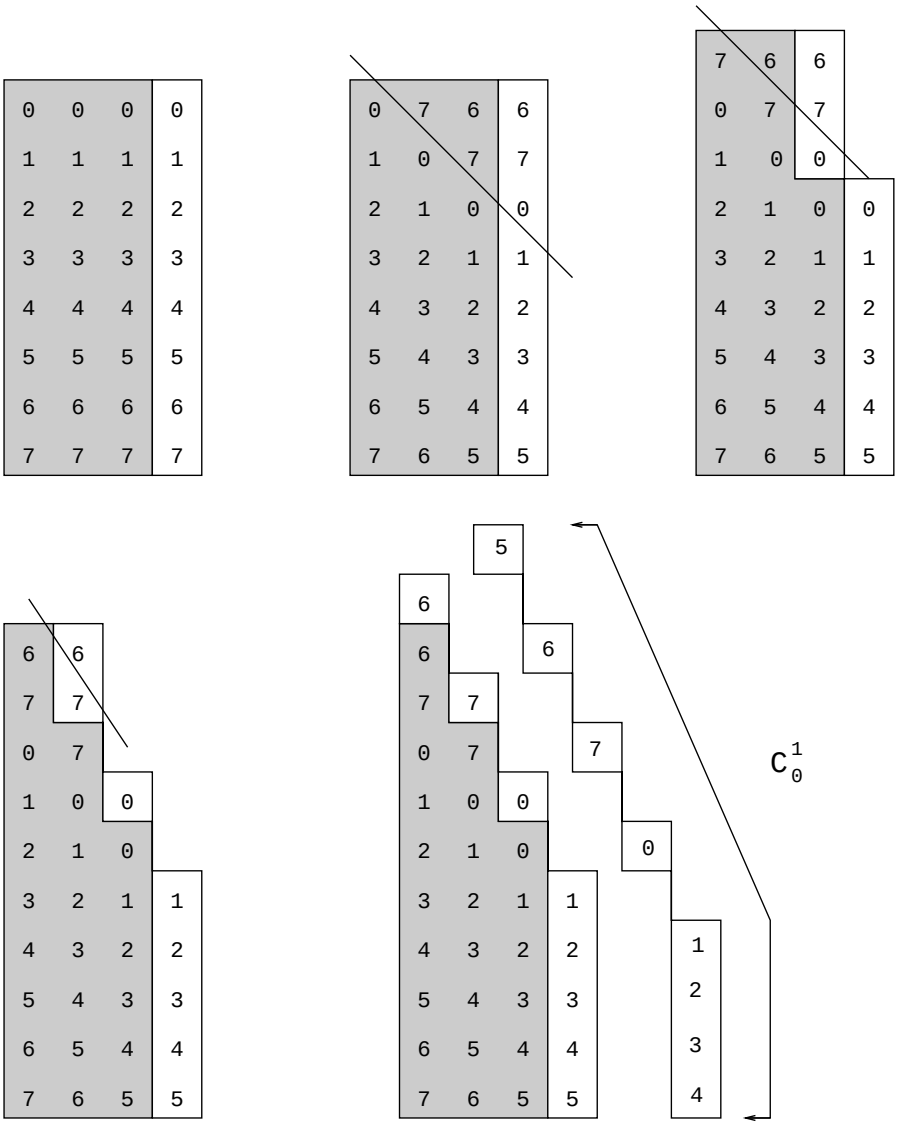


FIG. 5.2. *Creating a corner:* $h_g = 13$, $h_h = 8$, *load* 3.

around at only one place, which can increase the skew by no more than one over the minimum. □

Let w_c denote the smallest integer such that $h_g w_g \leq l h_h w_c$; i.e., w_c is the smallest width of a host of height h_h that can accommodate the given guest at the given load. Let k denote $h_g - h_h$.

THEOREM 5.4. *If* $h_g/2 < h_h < h_g$, *then the* (h_g, w_g)*-grid is embeddable in the* (h_h, w_h)*-cylinder, where* $w_h = (w_c + \lceil k/\lfloor l/2 \rfloor \rceil)$ *with dilation* 1 *and load* l.

Proof. We can square up the left end of the mapping as just demonstrated. The skew is an upper bound on the number of partially mapped columns at either end. Hence, by Lemma 5.3, the skew of the mapping at the right end is not greater than k. By Lemma 4.8, each host column covers at least $\lfloor l/2 \rfloor$ nodes from every row of the

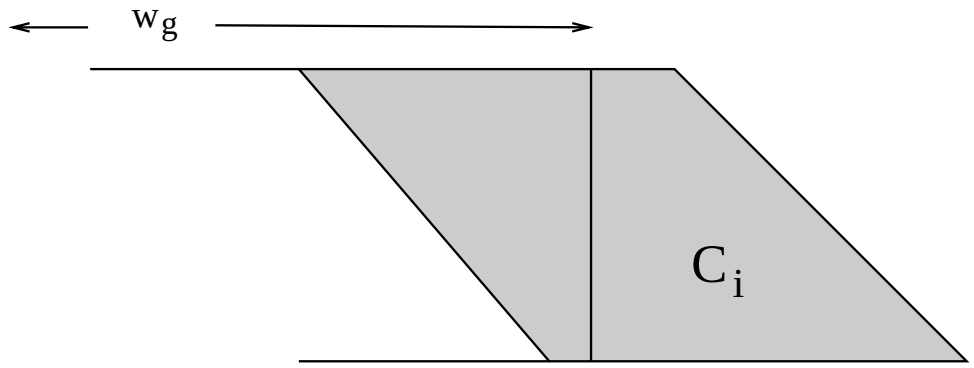


FIG. 5.3. *A high load can yield an optimal embedding.*

grid. Hence we need no more than $\lceil k/\lfloor l/2 \rfloor \rceil$ extra host columns to make up for the skew. □

We can also note that, if the load is sufficiently high, guest column w_g may be completely mapped, without the addition of extra host columns beyond column w_c. Such a situation is illustrated in Figure 5.3, where host column C_i is necessary but not completely used.

THEOREM 5.5. *If $lh_hw_h - h_gw_g \geq kh_h$ and $h_g/2 < h_h < h_g$, then the (h_g, w_g)-grid is embeddable in the (h_h, w_h)-cylinder with dilation* 1 *and load l.*

Proof. The skew is the number of partially mapped columns. The leftmost of these is of height $\leq h_g - 1$, and the next is of height $\leq h - 3$, etc., for $\leq k$ columns. Summing this series shows that the number of guest nodes in the partially filled columns is $\leq k(h_g - k) = kh_h$. Hence we need to add no extra host columns as long as $lh_hw_c - h_gw_g \geq kh_h$. □

6. Embedding into tori.

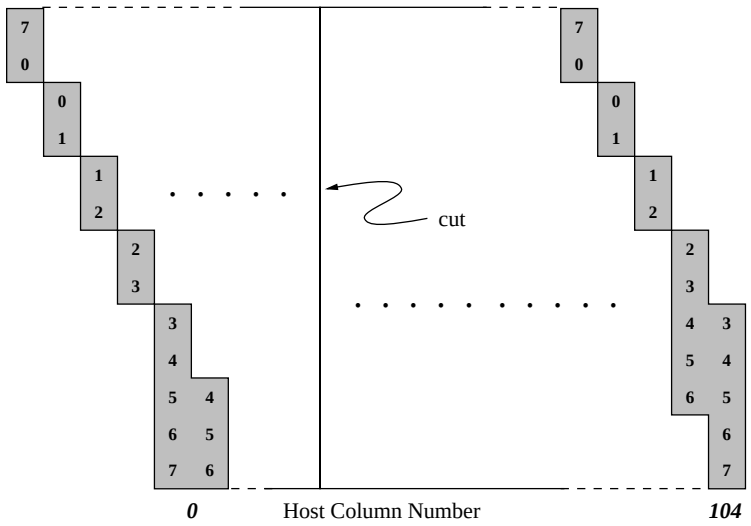
6.1. The problem. We now consider the "hardest" instances of the problem of embedding an (h_g, w_g)-grid into an (h_h, w_h)-torus, namely, those where w_g is maximal with respect to preserving $h_gw_g \leq lh_hw_h$. We will derive certain restrictions on the load and on the guest and host dimensions under which optimal solutions can be constructed.

To obtain these solutions, we match up the ends of a partial mapping. Consider first those special cases in which w_g is a multiple of the numerical periodicity (Definition 4.4 and Lemma 4.6). By definition, the host row numbers and profiles at the left and right ends of the partial mapping match. Hence we can cut the mapping vertically anywhere outside of the partially mapped columns, transpose the left and right portions of the mapping, and so produce a total mapping for the guest grid.

For example, consider Figure 6.1. In the illustration, a continuation of the example shown in Figure 4.1, $h_g = 13$, $h_h = 8$, and $l = 2$. Hence, by Lemma 4.6, the numerical periodicity is $p_n = 104$. So the figure is illustrating an embedding of the (13,128)-grid into the (8,104)-cylinder with load 2.

Of course, except in these very special cases, neither the row numbers nor the profiles at the ends of the mapping will match. We now show how to solve both of these problems.

6.2. Toroidal profiles. For embeddings into tori, we must initiate the partial mapping with a unique profile. We will refer to them here as *toroidal profiles*. In



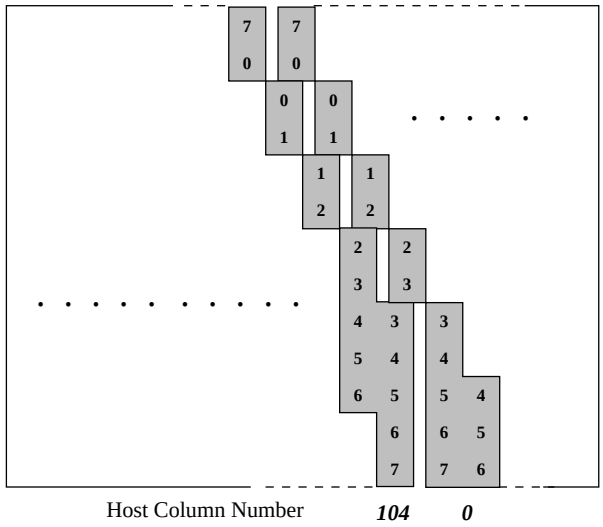


FIG. 6.1. *Matching up the ends of a partial mapping.*

[4, 5], where their existence and properties are demonstrated, they are called *Euclidean strings* because they are generated by an extension to Euclid's greatest common divisor algorithm. In this section, we show how to generate these special profiles and state their properties without repeating the proofs.

Let $\mathbf{p} = p_1 p_2 \cdots p_k$ denote a string of k nonnegative integers. Let $\rho(\mathbf{p})$ denote a left rotation of $\mathbf{p}$ by one position, i.e., $\rho(\mathbf{p}) = p_2 \cdots p_k p_1$, and let $\rho^d(\mathbf{p})$ denote a left rotation through d positions. Let $\sigma_i(\mathbf{p})$ be the string obtained from $\mathbf{p}$ by replacing p_i by $p_i + 1$ and $p_{(i+1) \bmod k}$ by $p_{(i+1) \bmod k} - 1$.

DEFINITION 6.1. *The string* $\boldsymbol{p}$ *is a* toroidal (k,n)-profile *if* $\sum_{i=1}^{k} p_i = n$, *and there exist integers* d *and* i *such that* $\sigma_i(\mathbf{p}) = \rho^d(\mathbf{p})$.

```
function profile (k, n : integer) : string;
if (k = 1) and (n = 1) then profile := 1
else if n > k then profile := expand(profile(k, n − k))
else profile := increment(profile(k − n, n));
```

FIG. 6.2. *Computing a profile.*

```
procedure parameters (k, n : integer);
if (k = 1) and (n = 1) then i := 0; j := 0; c := 0; d := 1
else if k < n then parameters (k, n − k); j := i + j; c := c + d  else parameters (k − n, n); i :=
i + j + 1; d := c + d;
```

FIG. 6.3. *Computing the profile parameters.*

In [4, 5], the following lemma is established.

LEMMA 6.2. *A toroidal (k, n)-profile exists iff $gcd(k, n) = 1$, where gcd denotes the greatest common divisor. If it exists, then it is unique.*

Suppose 1^r denotes a string of r ones, and suppose the function expand($\mathbf{p}$) replaces every element r in $\mathbf{p}$ by the string 01^r and the function increment($\mathbf{p}$) replaces every number r in $\mathbf{p}$ by $r + 1$. Then Figure 6.2 describes, in pseudocode, a procedure that computes a (k, n)-profile when k and n are relatively prime.

For the application of profiles to the computation of our embedding, we also need to compute the following parameters. We note that, since the profile is unique, all of these parameters are well defined.

1. i and d, as in the definition of profile,
2. j, the sum of elements p_1 through p_i,
3. c, the sum of the d elements to the left (cyclically) of and including p_i.

Figure 6.3 shows a pseudocode procedure which computes these parameters, assuming that all the variables are global. Figure 6.4 shows the computation of a string and its associated parameters for particular values of k and n.

We will refer to the parameters c and d as the *cost* and *displacement* of a profile, respectively, and we let $c_{k,n}$ and $d_{k,n}$ denote the cost and the displacement, respectively, of a toroidal (k, n)-profile.

LEMMA 6.3. *If $n + k \geq 2$, then $d_{k,n}n = c_{k,n}k + 1$.*

Proof. We argue by induction on $n + k$. For the basis, we note that, if $n + k = 2$, then $(n = 1)$ and $(k = 1)$, and hence the initialization defined by the algorithm satisfies the lemma.

We note that n and k are not equal because $gcd(n, k) = 1$. For the induction, suppose first that $n > k$. By the inductive hypothesis, $d_{k,n-k}(n - k) = c_{k,n-k}k + 1$. Hence $d_{k,n-k}n = d_{k,n-k}(n - k) + d_{k,n-k}k = (c_{k,n-k} + d_{k,n-k})k + 1$. However, when $n > k$, the algorithm sets $d_{k,n-k} = d_{k,n}$ and $c_{k,n-k} + d_{k,n-k} = c_{k,n}$. Hence $d_{k,n}n = c_{k,n}k + 1$.

Suppose $n < k$. By the inductive hypothesis, $d_{k-n,n}n = c_{k-n,n}(k - n) + 1$. Hence $(d_{k-n,n} + c_{k-n,n})n = c_{k-n,n}k + 1$. However, when $n < k$, the algorithm sets $d_{k-n,n} + c_{k-n,n} = d_{k,n}$ and $c_{k-n,n} = c_{k,n}$. Hence $d_{k,n}n = c_{k,n}k + 1$. ∎

So $d_{k,n}$ and $-c_{k,n}$ are in fact the constants computed by the standard extended Euclidean algorithm. Lemma 6.3 yields the following corollary.

COROLLARY 6.4. *$c_{k,n}$ is the multiplicative inverse of $(n - k)$ modulo k, i.e., $c_{k,n}(n - k) \equiv 1 \bmod n$.*

Proof. By Lemma 6.3, $d_{k,n}n = c_{k,n}k + 1$, implying $-c_{k,n}k = -d_{k,n}n + 1$. Hence $c_{k,n}(n-k) = c_{k,n}n-c_{k,n}k = c_{k,n}n-d_{k,n}n+1 = (c_{k,n}-d_{k,n})n+1$. Hence $c_{k,n}(n-k) \equiv$

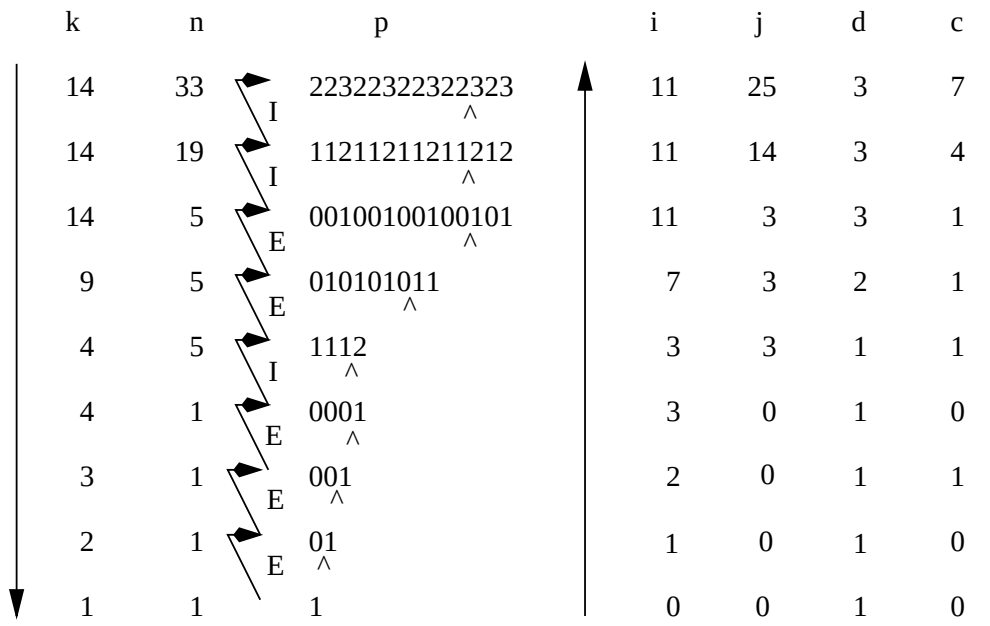


FIG. 6.4. *An example of computing a profile and its parameters.*

1 mod n. □

It is also shown in [4, 5] that all of the elements in a profile are of the form i or $i+1$ for some integer i. We will be using $(h_g - h_h, h_g)$-profiles, and so, since $h_h > h_g/2$, all of the profile elements will be ≥ 2.

6.3. Matching profiles. We are considering those problem instances in which $0 \leq l|H| - |G| < h_g$, the case in which the difference is zero being trivial. So we need a way of dropping between 1 and $h_g - 1$ host nodes from the mapping while simultaneously guaranteeing that the left profile of host column 0 is identical to the right profile of the last column.

Consider the running example, where $h_g = 13$ and $h_h = 8$. The application of the toroidal profile generator (see Figures 6.2 and 6.3) yields the profile 2 3 2-3 3, where swapping the hyphenated pair yields a rotation of the original string. In our application, the profile elements define the height of the steps in the left or right profiles of the host columns.

Suppose our mapping is based on a toroidal profile, and suppose we remove el-

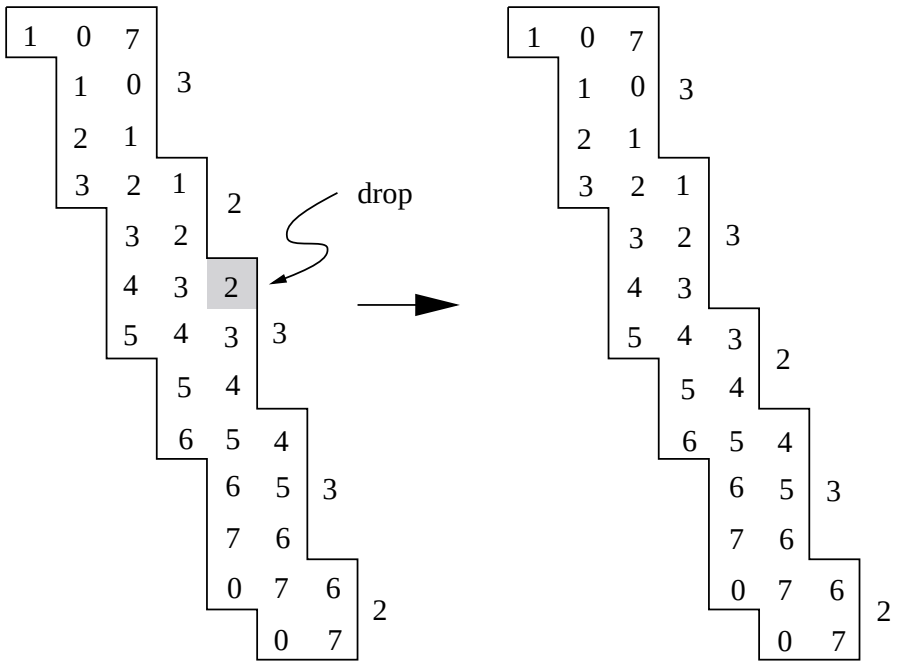


FIG. 6.5. *Dropping an element while preserving the profile.*

ement $(i+1)$ from the mapping in the right profile of some host column, where i is the profile parameter in Definition 6.1. Removing an element decreases the height of the step containing that element and increases the height of the step above. However, because the profile is toroidal, the resulting profile is a rotation of the original. Because the profile is preserved, except for a rotation, the operation can be repeated as many times as necessary. This is illustrated in Figure 6.5.

We deduce that we can drop $(l|H| - |G|) < h_g$ elements and be assured that the right profile of the rightmost host column is a rotation of the left profile of column 0. It remains only to show that not only are these two profiles rotations of each other, but they are identical. For the moment, let us assume that we apply the drop process to distinct host columns.

LEMMA 6.5. *If the drop operation is applied to $l|H| - |G|$ distinct host columns, the right profile of the rightmost host column is identical to the left profile of host column* 0.

Proof. Every host column rotates the profile l rows in what we will call the forward direction. Therefore, the profile shift s in the forward direction relative to its initial position is given by

$$s = (l(w_h \bmod p_s)) \bmod h_g, \tag{6.1}$$

where p_s denotes the structural periodicity. By Lemma 4.5, $p_s = h_g/gcd(l, h_g)$, and so we derive from (6.1)

$$s = (lw_h) \bmod h_g. \tag{6.2}$$

Let $\delta = lw_hh_h - w_gh_g$. Suppose we apply the drop process δ times. Each application shifts the profile c rows backward, where c is the cost of the profile. So the repeated

application shifts the profile $s' = \delta c \bmod h_g$ rows. However,

$$\delta c \bmod h_g = c(lw_h h_h - w_g h_g) \bmod h_g, \tag{6.3}$$

and so

$$s' = lw_h \bmod h_g = s \tag{6.4}$$

because $ch_h \bmod h_g = 1$ by Corollary 6.4. □

Hence the profile shift caused by the excess host columns is exactly compensated for by the repeated application of the drop process. Finally, we note that all of the drop operations can be applied to one host column, without removing all of the elements from that column from any one row.

LEMMA 6.6. *If $(l|H| - |G|) < h_g$ drop operations are applied to one host column and the load $l \geq 4$, the resulting width of the column is at least one.*

Proof. By Corollary 6.4, $ch_h \bmod h_g = 1$. It follows that c and h_g are relatively prime. Hence, since $\delta = (l|H| - |G|) < h_g$, if up to δ elements are dropped, they are all taken from different rows, so the width of the column is decreased by no more than one. By Lemma 4.8, the width of a column in the partial mapping is at least $\lfloor l/2 \rfloor$; hence, if $l \geq 4$, the resulting width is at least $\lfloor l/2 \rfloor - 1 = 1$. □

6.4. Matching row numbers. We describe a perturbation of the partial mapping which has the effect of incrementing all the host row numbers by one or two in all the columns following the perturbation. The perturbation can be applied to any instance of the problem for which the load is at least 4.

Let us suppose we are perturbing host column i. C_i is composed of C_i^0 through C_i^{l-1}. The perturbation simply adds one or two to all the host row values in C_i^2 and to those in C_i^3. If $l > 4$, then the mapping follows in the usual way after C_i^3. It is clear that the perturbation is still yielding dilation 1. In fact, at the junction where the perturbation occurs, adjacent guest nodes are mapped to the same host node; i.e., for those edges the dilation is zero. It is also clear that, since only the values in the mapping are changed, the profiles are not changed.

Figures 6.6 and 6.7 illustrate the process with a load 4 example, following from our running example. In these figures, the shading distinguishes C_i^0 and C_i^1 from C_i^2 and C_i^3.

Suppose we need to increment the row numbers by more than one. The process just described can be repeated on C_i^4 and C_i^5, etc., within a host column. Each column, therefore, can be used to obtain an increment of up to $2\lfloor (l-2)/2 \rfloor$. We observe that the row values never need to be incremented by more than $h_h - 1$ and that we can adjust by incrementing either by one or two. We have then justified the following lemma.

LEMMA 6.7. *If $\lceil (h_h - 1)/2 \rceil \leq w_h \lfloor (l-2)/2 \rfloor$ and the right profile of the last column in the mapping is identical to the left profile of column 0, then we can adjust row values so that they are equal in these two profiles.*

6.5. The solution. The computation of the total mapping can then proceed as follows.

1. Compute a toroidal $(h_g - h_h, h_g)$-profile together with the associated parameters using Figures 6.2 and 6.3.

2. Use this profile to initiate the computation of a partial mapping using Definition 4.1.

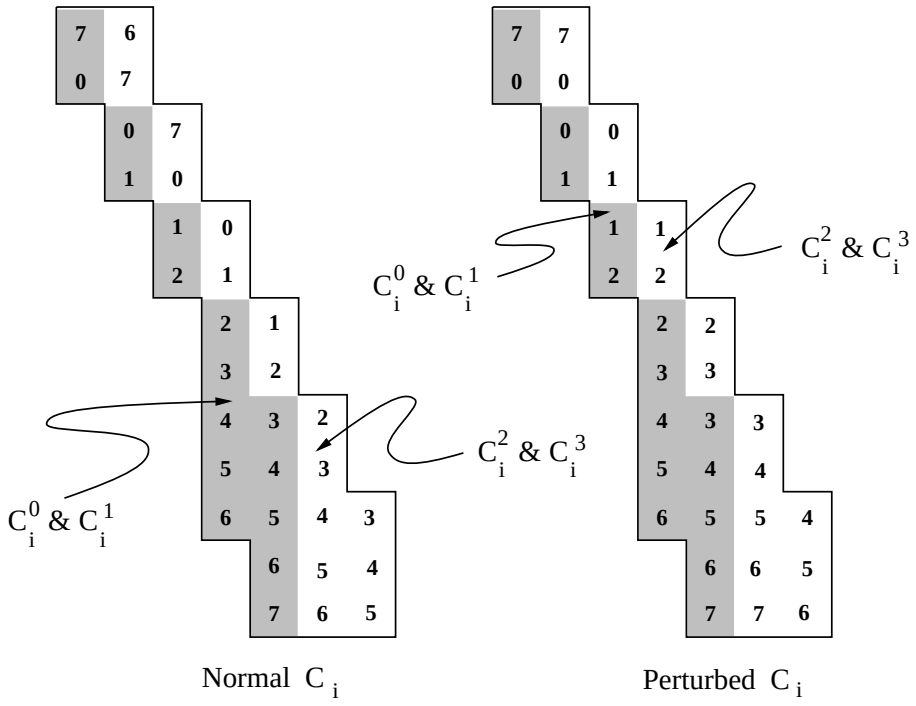


FIG. 6.6. *The perturbation that increments row numbers by one.*

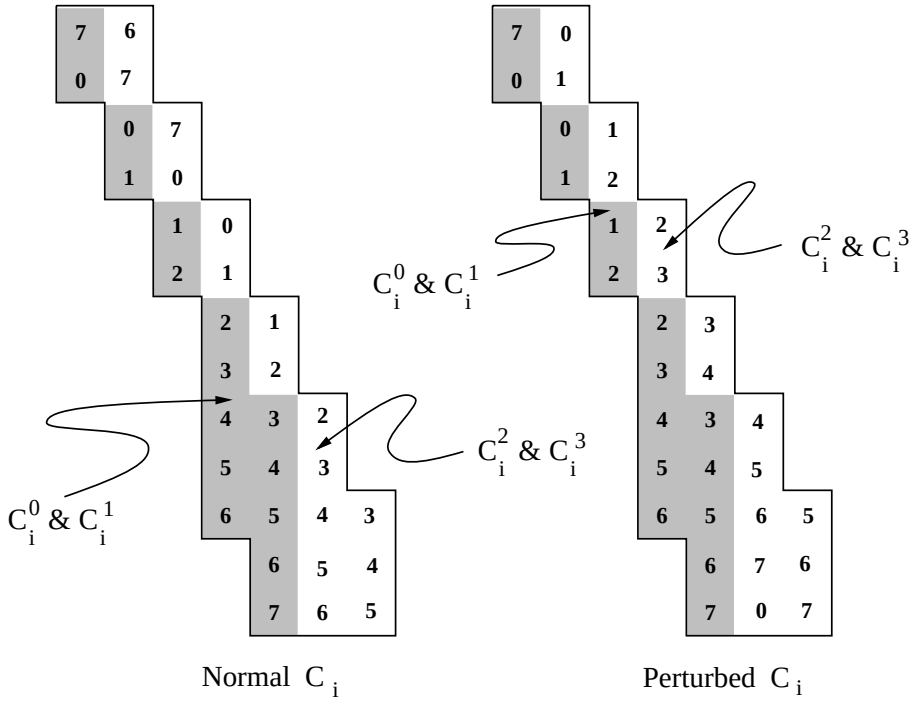


FIG. 6.7. *The perturbation that increments row numbers by two.*

3. Compute δ, the difference between available host nodes and the number of guest nodes, and apply the drop process δ times to (say) the right profile of the rightmost host column.

4. Compute the row number incrementation required, and apply the incrementation perturbation.

5. Cut the modified partial mapping vertically at any point outside of the partially mapped columns, and transpose the left and right portions produced by the cut.

The conditions under which this process is effective are summarized in the following theorem.

THEOREM 6.8. *If*

1. $h_g/2 < h_h < h_g$,
2. h_g *and* $(h_g - h_h)$ *are relatively prime,*
3. *the optimal load is at least* 4, *and*
4. $w_h(\lfloor (l-2)/2 \rfloor) \geq \lceil (h_h - 1)/2 \rceil + 1$,

then the (h_g, w_g)*-grid can be embedded in the* (h_h, w_h)*-torus with optimal load and dilation* 1.

Proof. The first item is a necessary condition for the construction of the partial mapping. The second is sufficient for a suitable profile to exist for the initiation of the mapping. The restriction on the load is sufficient for the increment operation and allows the drop operation to be repeated on one column.

The last item ensures that there is sufficient width within which to carry out both the row incrementation and the drop operations. Each host column i is composed of $\lfloor (l-2)/2 \rfloor$ pairs C_i^k, C_i^{k+1}, where $k \geq 2$. We may not assume that we can apply the increment operation on a pair that has already been the subject of the drop operation. As many as $\lfloor (h_h - 1)/2 \rfloor$ increment operations may be necessary. Consequently, we have sufficient host width if $w_h(\lfloor (l-2)/2 \rfloor) \geq \lfloor (h_h - 1)/2 \rfloor + 1$. □

7. Embedding into hypercubes. The previous section has shown that we can construct optimal embeddings into the torus under some restrictions on the load and dimensions of the guest and host. Because we can always find a torus of suitable height within a hypercube host, we obtain a much simpler result.

Consider the restrictions listed in Theorem 6.8 when applied to the embedding of the (h_g, w_g)-grid into the hypercube Q_d of dimension d.

1. By Lemma 3.2, there is a toroidal subgraph of the hypercube with height h_h in the range $h_g/2$ to h_g, namely, $h_h = 2^{\lfloor \log_2 h_g \rfloor}$, which is the next power of 2 less than h_g. Then also w_h is a power of two, namely, $2^{d - \lfloor \log_2 h_g \rfloor}$.

2. If h_g is an odd number, then h_g and h_h are relatively prime since h_h is now a power of two.

3. We retain the condition on the optimal load, namely, $l = \lceil h_g w_g / 2^d \rceil \geq 4$.

4. The last restriction cannot be dispensed with, but see the corollaries below. It can be simplified slightly by noting that h_h is a power of 2, yielding

$$w_h(\lfloor (l-2)/2 \rfloor) \geq h_h/2 + 1,$$

where h_h and w_h are now as defined in the first item.

Let h_h and w_h be the dimensions of the host torus as defined above as functions of the guest grid and host hypercube dimensions h_g, h_h, and d. Then the following theorems follow from Theorem 6.8.

THEOREM 7.1. *If*

1. h_g *is odd,*

2. *the optimal load* $l \geq 4$*, and*
3. $w_h(\lfloor (l-2)/2 \rfloor) \geq h_h/2 + 1$,

then the (h_g, w_g)*-grid can be embedded into* Q_d *with optimal load and dilation* 1.

We cannot guarantee that all instances of the problem present sufficient width to permit the necessary incrementation and drop operations. For example, the problem instance

$$h_g = 11, w_g = 14, h_h = 8, w_h = 4, l = 5$$

does not satisfy the third constraint of Theorem 7.1. Some slightly stronger but simpler forms of that constraint show more clearly that it is excluding only a small portion of problem instances.

COROLLARY 7.2. *If* h_g *is odd and the optimal load* $l \geq 4$ *and* $w_h \geq h_h$*, then the* (h_g, w_g)*-grid can be embedded into* Q_d *with optimal load and dilation* 1.

Proof. If $w_h \geq h_h$, then the third constraint of Theorem 7.1 is satisfied. ∎

COROLLARY 7.3. *If* h_g *is odd and either* $((l = 4$ *or* $l \geq 6)$ *and* $w_g \geq 2h_g)$ *or* $(l = 5$ *and* $w_g \geq 5h_g/2)$*, then the* (h_g, w_g)*-grid can be embedded into* Q_d *with optimal load and dilation* 1.

Proof. Since, by definition of the problem, $lh_hw_h \geq h_gw_g$ and $w_g \geq h_g > h_h$, if $w_g \geq 2h_g$, we deduce that $lw_h > 2h_h \geq 2h_h + 1$. However, $\lfloor l/2 \rfloor - 1 \geq l/4$ for $l = 4$ and $l \geq 6$, thus satisfying the third constraint. For load 5, the same algebra shows that $w_g \geq 5h_g/2$ suffices. ∎

The theorem implies that, if we group the problem instances by h_g, there are no more than a finite number of instances of the problem in each group that may not be solved by our method. Finally, we note that instances where h_g is even either are trivial if h_g is a power of two or can be reduced to instances where h_g is odd.

THEOREM 7.4. *If* h_g *is of the form* $a2^k$ *for some odd integer* a *and positive integer* k *and either* $k \leq d$ *and there is an optimal embedding from the* (a, w_g)*-grid into the hypercube of degree* $d - k$ *or* $k > d$*, then the* (h_g, w_g)*-grid can be embedded into the hypercube of degree* d *with optimal load and dilation* 1.

Proof. If $k \leq d$, we note that it must be that $2^k | (l2^d - a2^k w_g)$. Hence we can solve the subproblem of embedding the (a, w_g)-grid into Q_{d-k} with load l and then construct a solution to the original problem by simple duplication of the host.

If $k > d$, then $|Q_d| < h_g$ and $|Q_d| \mid h_g$. Hence an embedding can be constructed by simple duplication of the host. ∎

THEOREM 7.5. *If* h_g *is a power of two, then an optimal embedding always exists into any hypercube.*

Proof. If $h_g = 2^k$ and $d \leq k$, then the host is not bigger than one column of the guest. Hence a simple duplication technique can cover the guest with copies of the host.

If $h_g = 2^k$ and $d > k$, then we consider the $(2^k, 2^{d-k})$-grid, which is a subgraph of the host. A simple duplication technique can then cover the guest with copies of the host. ∎

Theorems 7.1, 7.4, and 7.5 cover all possibilities for the form of h_g and constitute the main result of this paper. Corollaries 7.2 and 7.3 slightly simplify the third condition.

8. Conclusions. We have described a novel grid to hypercube embedding construction which is simple and optimal with respect to load and dilation for all except perhaps finitely many instances for each grid height, when the load is at least four.

The method also yields optimal embeddings into the torus, under constraints on the relative heights of the torus and grid. For embeddings into the cylinder, the same constraints are necessary, and we may have to extend the cylinder somewhat beyond the minimum necessary to accommodate the grid.

Open problems include the removal of the restrictions on load 4 and above hypercube embeddings and the question as to whether optimal load dilation 1 hypercube embeddings exist for loads 2 and 3. It may also be possible to push the results we have obtained here further for cylindrical and toroidal hosts.

Acknowledgment. We are indebted to the referees for thorough readings and for their many valuable suggestions and corrections.

REFERENCES

[1] S. Bettayeb, Z. Miller, and I. H. Sudborough, *Embedding grids into hypercubes*, J. Comput. System Sci., 45 (1992), pp. 340–366.

[2] M. Y. Chan, *Embedding of grids into optimal hypercubes*, SIAM J. Comput., 20 (1991), pp. 834–864.

[3] M. Y. Chan, F. Chin, C. N. Chu, and W. K. Mak, *Dilation-5 embedding of 3-dimensional grids into hypercubes*, J. Parallel Distrib. Comput., 33 (1996), pp. 98–106.

[4] J. Ellis, F. Ruskey, and J. Sawada, *Euclidean strings*, in Proceedings of the 11th Australasian Workshop on Combinatorial Algorithms, University of Newcastle, Australia, 2000, pp. 87–92.

[5] J. Ellis, F. Ruskey, J. Sawada, and J. Simpson, *Euclidean strings*, Theoret. Comput. Sci., to appear.

[6] J. A. Ellis, *Embedding rectangular grids into square grids*, IEEE Trans. Comput., 40 (1991), pp. 46–52.

[7] J. A. Ellis, *Embedding grids into grids: Techniques for large compression ratios*, Networks, 27 (1996), pp. 1–17.

[8] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan-Kaufmann, San Mateo, CA, 1992.

[9] R. G. Melhem and G. Y. Hwang, *Embedding rectangular grids into square grids with dilation two*, IEEE Trans. Comput., 39 (1990), pp. 1446–1455.

[10] Z. Miller and I. H. Sudborough, *Compressing grids into small hypercubes*, Networks, 24 (1994), pp. 327–358.

[11] M. Röttger, U.-P. Schroeder, and W. Unger, *Embedding 3-dimensional grids into optimal hypercubes*, in Parallel and Distributed Computing, Lecture Notes in Comput. Sci. 805, Springer-Verlag, Berlin, 1994, pp. 81–94.

[12] M. Röttger and U.-P. Schroeder, *Efficient embeddings of grids into grids*, Discrete Appl. Math., 108 (2001), pp. 143–173.

[13] M. Röttger and U.-P. Schroeder, *Embedding 2-dimensional grids into optimal hypercubes with edge-congestion 1 or 2*, Parallel Process. Lett., 8 (1998), pp. 231–242.

[14] F. C. Sang and I. H. Sudborough, *Embedding large meshes into small ones*, in Proceedings of the IEEE International Conference on Circuits and Systems, New Orleans, LA, 1990, pp. 323–326.

[15] H. L. Shou-Hsuan S. Huang and R. M. Verma, *A new combinatorial approach to optimal embeddings of rectangles*, Algorithmica, 16 (1996), pp. 161–180.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 408–434

COMMON-FACE EMBEDDINGS OF PLANAR GRAPHS*

ZHI-ZHONG CHEN†, XIN HE‡, AND MING-YANG KAO§

Abstract. Given a planar graph $\mathcal{G}$ and a sequence $\mathcal{C}_1, \ldots, \mathcal{C}_q$, where each $\mathcal{C}_i$ is a family of vertex subsets of $\mathcal{G}$, we wish to find a plane embedding of $\mathcal{G}$, if any exists, such that, for each $i \in \{1, \ldots, q\}$, there is a face F_i in the embedding whose boundary contains at least one vertex from each set in $\mathcal{C}_i$. This problem has applications in the recovery of topological information from geographical data and the design of constrained layouts in VLSI. Let I be the input size, i.e., the total number of vertices and edges in $\mathcal{G}$ and the families $\mathcal{C}_i$, counting multiplicity. We show that this problem is NP-complete in general. We also show that it is solvable in $O(I \log I)$ time for the special case in which, for each input family $\mathcal{C}_i$, each set in $\mathcal{C}_i$ induces a connected subgraph of the input graph $\mathcal{G}$. Note that the classical problem of simply finding a planar embedding is a further special case of this case with $q = 0$. Therefore, the processing of the additional constraints $\mathcal{C}_1, \ldots, \mathcal{C}_q$ incurs only a logarithmic factor of overhead.

Key words. planar graph, planar embedding, topological inference

AMS subject classifications. 05C85, 68R05, 68R10, 68W40

PII. S009753970037775X

1. Introduction.

It is a fundamental problem in mathematics (see, e.g., [13, 17, 18, 19, 20, 29]) to embed a graph into a given surface while optimizing certain objectives required by applications. (Throughout this paper, a graph may have multiple edges and self-loops, but a simple graph has neither.) A graph is *planar* if it can be embedded on the plane so that any pair of edges intersect only at their endpoints; a *plane* graph is a planar graph together with such an embedding. A classical variant of the problem is to test whether a given graph is planar and, in case it is, to find a planar embedding. This planarity problem can be solved in linear time sequentially [4, 5, 19] and efficiently in parallel [26].

In this paper, we initiate the study of the following new planarity problem. Let $\mathcal{G}$ be a planar graph. Let $\mathcal{M}$ be a sequence $\mathcal{C}_1, \ldots, \mathcal{C}_q$, where each $\mathcal{C}_i$ is a family of vertex subsets of $\mathcal{G}$. A plane embedding Φ of $\mathcal{G}$ *satisfies* $\mathcal{C}_i$ if the boundary of some face in Φ contains at least one vertex from each set in $\mathcal{C}_i$. Φ *satisfies* $\mathcal{M}$ if it satisfies all $\mathcal{C}_i$. $\mathcal{G}$ *satisfies* $\mathcal{M}$ if $\mathcal{G}$ has an embedding that satisfies $\mathcal{M}$.

PROBLEM 1 (the common-face embedding (CFE) problem).

- *Input*: A planar graph $\mathcal{G}$ and a sequence $\mathcal{M}$ of families of vertex subsets of $\mathcal{G}$.
- *Question*: Does $\mathcal{G}$ satisfy $\mathcal{M}$?

Let I be the input size, i.e., the total number of vertices and edges in $\mathcal{G}$ and the families $\mathcal{C}_i$, counting multiplicity. We first show that the CFE problem is NP-complete in general. Then, for the special case in which each vertex subset in each $\mathcal{C}_i$

*Received by the editors September 11, 2000; accepted for publication (in revised form) October 31, 2002; published electronically February 4, 2003. A preliminary form of this paper appeared in *Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms,* 1999, pp. 195–204.
http://www.siam.org/journals/sicomp/32-2/37775.html

†Department of Mathematical Sciences, Tokyo Denki University, Hatoyama, Saitama 350-0394, Japan (chen@r.dendai.ac.jp).

‡Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY 14260 (xinhe@cse.buffalo.edu). This author's research was supported in part by NSF grant CCR-9912418.

§Department of Computer Science, Northwestern University, 1890 Maple Avenue, Evanston, IL 60201 (kao@cs.northwestern.edu). This author's research was supported in part by NSF grant CCR-9531028.

induces a connected subgraph of $\mathcal{G}$, we give an $O(I \log I)$-time algorithm which finds a plane embedding satisfying $\mathcal{M}$, if any exists. Note that the classical problem of simply finding a planar embedding is a further special case of this special case with $q = 0$. Therefore, the processing of the additional constraints $\mathcal{C}_1, \ldots, \mathcal{C}_q$ incurs only a logarithmic factor of overhead.

The CFE problem arises naturally from topological inference [6]. For instance, in the conference version of this paper [7], a less general and less efficient variant of our algorithm for the special case was employed to design fast algorithms for reconstructing maps from scrambled partial data in geometric information systems [7]. In this application [8, 9, 10, 15, 23, 24], each vertex subset in $\mathcal{M}$ describes a recognizable geographical feature, and each face in a planar embedding represents a geographical region. Each family in $\mathcal{M}$ is a set of features that are known to be near each other, i.e., surrounding the same region (on the boundary of the same face). Similarly, our algorithm for the special case can compute a constrained layout of VLSI modules [14], where each vertex subset consists of the ports of a module, and each subset family specifies a set of modules that are required to be close to each other [7].

To the best of our knowledge, the conference version of this paper is the first to investigate the CFE problem [7]. A related problem has been studied in the context of speeding up the computation of Steiner trees and minimum-concave-cost network flows [11, 25, 3]. Given a planar graph $G = (V, E)$ and a set of special vertices $S \subseteq V$, the pair (G, S) is called *k-planar* if all the vertices in S are on the boundaries of at most k faces of a planar embedding of G. Bienstock and Monma [3] showed that testing k-planarity is NP-complete if k is part of the input but takes linear time for any fixed k.

The remainder of this paper is organized as follows. Section 2 proves the NP-completeness result and formally states the main theorem on the CFE algorithm (Theorem 2.2). Sections 3 through 6 prove the main theorem by detailing the algorithm for the key cases in which $\mathcal{G}$ is (1) triconnected, (2) disconnected, (3) connected, or (4) biconnected, respectively. The triconnected case is the base case in that the other cases are eventually reduced to it. For this reason, this case is analyzed before the other cases. Section 7 concludes this paper with some directions for further research.

2. Basics and the main results.

2.1. Basic definitions. Let G be a graph. $|G|$ denotes the *size* of G, i.e., the total number of vertices and edges in G. $\mathcal{V}(G)$ denotes the vertex set of G. If G is a plane graph, then $\mathcal{F}(G)$ denotes the set of faces of G.

A set U is *G-local* if $U \subseteq \mathcal{V}(G)$. A family $\mathcal{C}$ of sets is *G-local* if every set in $\mathcal{C}$ is G-local.

For a subset U of $\mathcal{V}(G)$, the *subgraph of G induced by U* is the graph (U, E_U), where E_U consists of all edges e of G whose endpoints both belong to U; $G - U$ denotes the subgraph of G induced by $\mathcal{V}(G) - U$.

A *cut* vertex of G is one whose removal increases the number of connected components in G; a *block* of G is a maximal subgraph of G with no cut vertex. Let $\Psi(G)$ denote the forest whose vertices are the cut vertices and the blocks of G and whose edges are those $\{v, B\}$ such that v is a cut vertex of G, B is a block of G, and $v \in \mathcal{V}(B)$. Note that $\Psi(G)$ is a tree if G is connected. A *block vertex* of $\Psi(G)$ is a vertex of $\Psi(G)$ that is a block of G.

Let w be a cut vertex of G. Let $W_1, \ldots, W_k$ be the vertex sets of the connected components of $G - \{w\}$. Let G_i be the subgraph of G induced by $\{w\} \cup W_i$.

$G_1, \ldots, G_k$ are called the *augmented components* of G induced by w.

G is *biconnected* if it is connected and has at least two vertices but no cut vertex. G is triconnected if it is biconnected and has at least three vertices and the removal of any two vertices does not disconnect it.

The *size* of a set S, denoted by $|S|$, is the number of elements in S. The *size* of a family $\mathcal{C}$ of sets, denoted by $|\mathcal{C}|$, is $\sum_S |S|$, where S ranges over all sets in $\mathcal{C}$. The *size* of a sequence $\mathcal{M}$ of families of sets, denoted by $|\mathcal{M}|$, is $\sum_{\mathcal{C}} |\mathcal{C}|$, where $\mathcal{C}$ ranges over all families in $\mathcal{M}$.

2.2. An NP-completeness result.

THEOREM 2.1. *The CFE problem is NP-complete.*

Proof. We reduce the SATISFIABILITY problem [14] to the CFE problem. Let ϕ be a CNF formula over variables $x_1, \ldots, x_n$ with $n \geq 2$. Let $C_1, \ldots, C_m$ be the clauses of ϕ, each regarded as the set of literals in it. We construct a simple biconnected planar graph $\mathcal{G} = (V_1 \cup V_2, E)$ as follows. $V_1 = \{x_1, \ldots, x_n\} \cup \{\bar{x}_1, \ldots, \bar{x}_n\} \cup \{c_1, \ldots, c_m\}$. $V_2 = \{u_0, \ldots, u_n\}$. For each x_i, $\mathcal{G}$ contains edges $\{u_{i-1}, x_i\}$, $\{x_i, u_i\}$, $\{u_{i-1}, \bar{x}_i\}$, $\{\bar{x}_i, u_i\}$. The only other edges of $\mathcal{G}$ are $\{u_0, c_1\}$, $\{c_1, c_2\}$, $\{c_2, c_3\}$, $\ldots$, $\{c_{m-1}, c_m\}$, $\{c_m, u_n\}$, $\{u_n, u_0\}$. Let $\mathcal{M}$ be the sequence $\{\{c_1\}, C_1\}, \ldots, \{\{c_m\}, C_m\}$. Observe that, in every plane embedding Φ of $\mathcal{G}$, the cycle $c_1, \ldots, c_m, u_n, u_0$ forms the boundary of some face F. Moreover, for each $i = 1, \ldots, n$, exactly one of x_i and $\bar{x}_i$ is on the boundary of the face other than F whose boundary contains the path $c_1, \ldots, c_m$. Also, for every set $S \subseteq \{x_1, \ldots, x_n\} \cup \{\bar{x}_1, \ldots, \bar{x}_n\}$ with $|S \cap \{x_i, \bar{x}_i\}| = 1$ for all $i = 1, \ldots, n$, $\mathcal{G}$ has a plane embedding where the boundary of some face contains the path $c_1, \ldots, c_m$ and the vertices in S. Therefore, ϕ is satisfiable if and only if $\mathcal{G}$ satisfies $\mathcal{M}$. □

2.3. The main theorem. Although the input to the CFE problem is a planar graph $\mathcal{G}$, it is easy to see that $\mathcal{G}$ satisfies a given sequence $\mathcal{M}$ if and only if its underlying simple graph (i.e., the simple graph obtained from $\mathcal{G}$ by deleting multiple edges and self-loops) satisfies the same $\mathcal{M}$. Thus, throughout the rest of this paper, unless explicitly stated otherwise, $\mathcal{G}$ and $\mathcal{M}$ always denote the input simple graph and the input sequence to our algorithm for the CFE problem, respectively. Also, I always denotes $|\mathcal{G}| + |\mathcal{M}|$, i.e., the size of the input to our algorithm.

The next theorem is the main theorem of this paper. In light of this theorem, the remainder of the paper assumes that every vertex subset of $\mathcal{G}$ in $\mathcal{M}$ induces a connected subgraph of $\mathcal{G}$.

THEOREM 2.2. *If every vertex subset in $\mathcal{M}$ induces a connected subgraph of $\mathcal{G}$, then the CFE problem can be solved in $O(I \log I)$ time.*

Proof. We consider three special cases:

- *Case* M1. $\mathcal{G}$ is connected.
- *Case* M2. $\mathcal{G}$ is biconnected.
- *Case* M3. $\mathcal{G}$ is triconnected.

In section 3, Theorem 3.8 solves Case M3 of the CFE problem faster than the desired time bound. In section 4, Theorem 4.3 reduces this theorem to Case M1. In section 5, Theorem 5.3 reduces Case M1 to Case M2. In section 6, Theorem 6.1 uses Theorem 3.8 to solve Case M2 of the CFE problem within the desired running time. This theorem follows from Theorems 4.3, 5.3, and 6.1. □

As mentioned in section 1, Case M3 is the base case, meaning that the other cases are eventually reduced to it. So the next section describes an algorithm for this case.

3. Solving Case M3, where $\mathcal{G}$ is triconnected. Throughout this section, we assume that $\mathcal{G}$ is triconnected. Then $\mathcal{G}$ has a unique combinatorial embedding up to the choice of the exterior face [21, 30]. Thus the CFE problem reduces in linear time to that of finding all the faces in the embedding whose boundaries intersect every set in some $\mathcal{C}_i$. The naive algorithm takes $\Theta(|\mathcal{G}||\mathcal{M}|)$ time. We solve the latter problem more efficiently by recursively solving Problem 2 defined below.

Throughout this section, for technical convenience, the vertices of a plane graph are indexed by distinct positive integers. The faces are indexed by positive integers or -1. The faces indexed by positive integers have distinct indices and are called the *positive* faces. Those indexed by -1 are the *negative* faces.

Let $\mathcal{H}$ be a plane graph. A *vf-set* of $\mathcal{H}$ is a set of vertices and positive faces in $\mathcal{H}$. A *vf-family* of $\mathcal{H}$ is a family of vf-sets of $\mathcal{H}$. A *vf-sequence* of $\mathcal{H}$ is a sequence of vf-families of $\mathcal{H}$. For a vf-family $\mathcal{D} = \{S_1, \ldots, S_d\}$ of $\mathcal{H}$, we define $\Lambda_{\mathrm{f}}(\mathcal{H}, \mathcal{D})$ and $\mathrm{ACF}(\mathcal{H}, \mathcal{D})$ as follows:

1. $\Lambda_{\mathrm{v}}(\mathcal{H}, \mathcal{D}) = \cap_{i=1}^{d} S_i \cap \mathcal{V}(\mathcal{H})$.
2. $\Lambda_{\mathrm{f}}(\mathcal{H}, \mathcal{D})$ is the set of positive faces F of $\mathcal{H}$ such that, for each $S_i \in \mathcal{D}$, F is a face in S_i or its boundary intersects $S_i - \Lambda_{\mathrm{v}}(\mathcal{H}, \mathcal{D})$.
3. $\mathrm{ACF}(\mathcal{H}, \mathcal{D}) = \Lambda_{\mathrm{v}}(\mathcal{H}, \mathcal{D}) \cup \Lambda_{\mathrm{f}}(\mathcal{H}, \mathcal{D})$.

PROBLEM 2 (the all-common-face (ACF) problem).

- *Input*: A plane graph $\mathcal{H}$ and a vf-sequence $\mathcal{N}$ of $\mathcal{H}$.
- *Output*: $\mathrm{ACF}(\mathcal{H}, \mathcal{D}_1), \ldots, \mathrm{ACF}(\mathcal{H}, \mathcal{D}_q)$, where $\mathcal{D}_1, \ldots, \mathcal{D}_q$ are the vf-families in $\mathcal{N}$.

Throughout the rest of this section, $\mathcal{H}$ and $\mathcal{N}$ always denote the input graph and the input sequence to our algorithm for the ACF problem, respectively.

To solve the ACF problem recursively, $\mathcal{H}$ need not be simple or triconnected. Furthermore, those faces that are indexed by -1 are ruled out as final output during recursions. To solve the problem efficiently, each vertex in $\Lambda_{\mathrm{v}}(\mathcal{H}, \mathcal{D}_i)$ is meant as a succinct representation of all the faces whose boundaries contain that vertex. Similarly, the positive faces in the input $\mathcal{D}_i$ and the output are represented by their indices.

The next observation relates the CFE problem and the ACF problem.

OBSERVATION 3.1. *Let the faces of $\mathcal{G}$ be indexed by positive integers. Then the output to the CFE problem is "yes" if and only if, for all $\mathcal{C}_i$, $\mathrm{ACF}(\mathcal{G}, \mathcal{C}_i) \neq \emptyset$.*

Section 3.1 proves a counting lemma useful for analyzing the time complexity of our algorithms for the ACF problem. Section 3.2 provides a technique for simplifying $\mathcal{H}$ during recursions. Section 3.3 uses this technique to recursively solve the ACF problem without increasing the total size of the subproblems.

3.1. A counting lemma.

LEMMA 3.2.

1. *Let v_1 and v_2 be distinct vertices in $\mathcal{G}$. Let F_1 and F_2 be distinct faces in $\mathcal{G}$. Then both v_1 and v_2 are on the boundaries of both F_1 and F_2 if and only if v_1 and v_2 form a boundary edge of both F_1 and F_2.*
2. *Given a set U of vertices in $\mathcal{G}$, there are $O(|U|)$ faces in $\mathcal{G}$ whose boundaries each contain at least two vertices in U.*
3. *Given a set $\mathcal{P}$ of faces in $\mathcal{G}$, there are $O(|\mathcal{P}|)$ vertices in $\mathcal{G}$ which are each on the boundaries of at least two faces in $\mathcal{P}$.*

Proof. We prove the statements separately as follows.

Statement 1. This statement immediately follows from the condition that $\mathcal{G}$ is triconnected with no multiple edges.

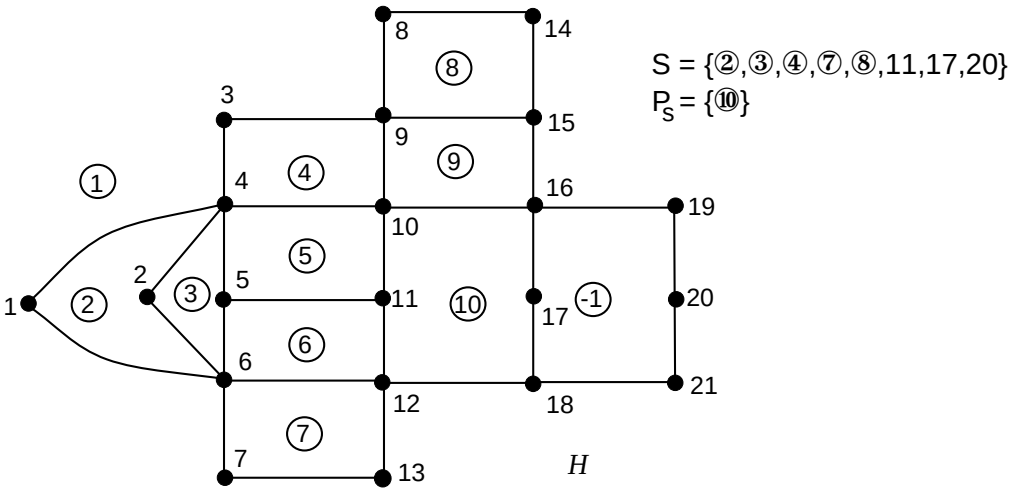


FIG. 3.1. *This is an example of a graph $\mathcal{H}$, a vf-set S, and $\mathcal{P}_S$, where a number in a circle is the index of the corresponding face.*

Statement 2. Since $\mathcal{G}$ has no multiple edges, $\mathcal{G}$ contains $O(|U|)$ edges between distinct vertices in U. Then this statement follows from statement 1 and the fact that an edge in a simple plane graph can be a boundary edge of at most two faces.

Statement 3. If $\mathcal{G}$ has at most three vertices, the statement holds trivially. Otherwise, the statement follows from statement 2 and the fact that the dual of $\mathcal{G}$ is also a simple triconnected plane graph [22]. □

COROLLARY 3.3. *If $\mathcal{H}$ is simple and triconnected, then the output of the ACF problem has size $O(|\mathcal{N}|)$.*

Proof. This corollary follows from Lemma 3.2(2). □

3.2. Simplifying $\mathcal{H}$ over a vf-set. To solve the ACF problem efficiently, we simplify the input graph $\mathcal{H}$ by removing unnecessary edges and vertices as follows.

For a vf-set S of $\mathcal{H}$, the plane graph $\mathcal{H}\diamondsuit S$ of $\mathcal{H}$ constructed as follows is said to *simplify* $\mathcal{H}$ over S. An example is illustrated in Figures 3.1, 3.2, and 3.3.

Let $\mathcal{P}_S$ be the set of the positive faces in $\mathcal{H}$ whose boundaries each contain at least two distinct vertices in $S \cap \mathcal{V}(\mathcal{H})$. Let $\mathcal{H}_S$ be the plane subgraph of $\mathcal{H}$ (1) whose vertices are those in $S \cap \mathcal{V}(\mathcal{H})$ and the boundary vertices of the faces in $(S \cap \mathcal{F}(\mathcal{H})) \cup \mathcal{P}_S$ and (2) whose edges are the boundary edges of the faces in $(S \cap \mathcal{F}(\mathcal{H})) \cup \mathcal{P}_S$. Note that $\mathcal{H}_S$ inherits a plane embedding from $\mathcal{H}$.

Let U_3 be the set of vertices which are of degree at least 3 in $\mathcal{H}_S$; note that each vertex in U_3 appears on the boundaries of at least two faces in $(S \cap \mathcal{F}(\mathcal{H})) \cup \mathcal{P}_S$. A *compressible* path P in $\mathcal{H}_S$ is a maximal path, which may be a cycle, such that (1) every internal vertex of P appears only once in it, and (2) no internal vertex of P is in $S \cup U_3$. Note that, by the choice of U_3, every internal vertex of a compressible path is of degree 2 in $\mathcal{H}_S$. We use this property to further simplify $\mathcal{H}_S$. Let $\mathcal{H}\diamondsuit S$ be the plane graph obtained from $\mathcal{H}_S$ by replacing each compressible path with an edge between its endpoints. This edge is embedded by the same curve in the plane as the path is. For technical consistency, if a compressible path forms a cycle and its endpoint is not in $S \cup U_3$, then we replace it with a self-loop for the vertex of the cycle with the smallest index.

Each vertex in $\mathcal{H}\diamondsuit S$ is given the same index as in $\mathcal{H}$. Note that the closure of

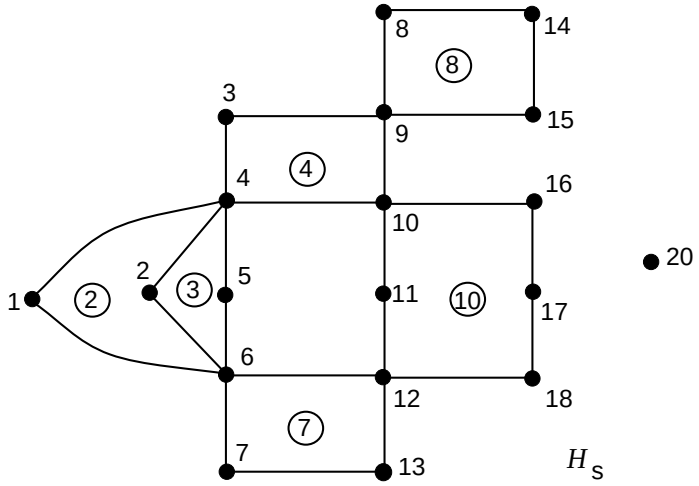


FIG. 3.2. *This is the graph $\mathcal{H}_S$ for the example of $\mathcal{H}$ and S in Figure* 3.1.

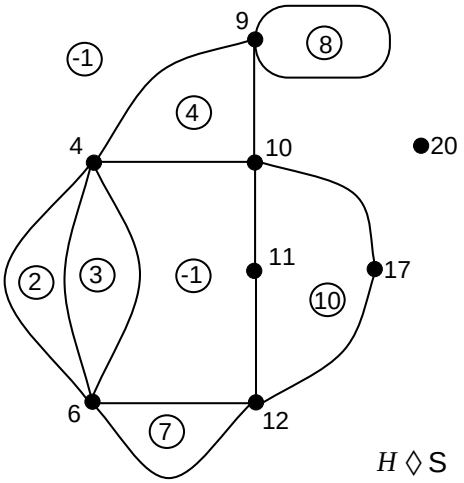


FIG. 3.3. *This is the graph $\mathcal{H}\diamondsuit S$ for the example of $\mathcal{H}$ and S in Figure* 3.1.

the interior of each face of $\mathcal{H}\diamondsuit S$ is the union of those of several faces or just one in $\mathcal{H}$. Let F be a face in $\mathcal{H}\diamondsuit S$, and let F' be one in $\mathcal{H}$. Let σ (respectively, σ') denote the closure of the interior of F (respectively, F'). If $\sigma = \sigma'$, then F and F' are regarded as the same face, and F is assigned the same index in $\mathcal{H}\diamondsuit S$ as F' is in $\mathcal{H}$. For technical conciseness, these two faces are identified with each other. If σ is the union of the closures of the interiors of two or more faces in $\mathcal{H}$, F is not the same as any face in $\mathcal{H}$ and is indexed by -1. This completes the definition of $\mathcal{H}\diamondsuit S$.

LEMMA 3.4.

1. *Given $\mathcal{H}$ and S, we can compute $\mathcal{H}\diamondsuit S$ in $O(|\mathcal{H}| + |S|)$ time.*
2. *Let S' be a vf-set of $\mathcal{H}\diamondsuit S$. If $S' \subseteq S$, then $\mathcal{H}\diamondsuit S' = (\mathcal{H}\diamondsuit S)\diamondsuit S'$.*
3. *If $\mathcal{H}$ simplifies $\mathcal{G}$ over a vf-set S^* with $S \subseteq S^*$, then $|\mathcal{H}\diamondsuit S| = O(|S|)$.*

Proof. Statements 1 and 2 are straightforward. To prove statement 3, it suffices to prove $|\mathcal{G}\diamondsuit S| = O(|S|)$ since by statement 2, $\mathcal{H}\diamondsuit S = \mathcal{G}\diamondsuit S$.

To bound the number of vertices in $\mathcal{G}\diamondsuit S$, let $\mathcal{P}_S$ and U_3 be as specified in the definition of $\mathcal{G}\diamondsuit S$. Let U_1 be the set of vertices v in $\mathcal{G}\diamondsuit S$ such that v appears on the boundary of exactly one face in $(S \cap \mathcal{F}(\mathcal{G})) \cup \mathcal{P}_S$. Then $(S \cap \mathcal{V}(\mathcal{G})) \cup U_3 \cup U_1$ consists of all the vertices in $\mathcal{G}\diamondsuit S$. Note that $|U_1| \leq |(S \cap \mathcal{F}(\mathcal{G})) \cup \mathcal{P}_S|$. Also, by Lemma 3.2(3), $|U_3| = O(|(S \cap \mathcal{F}(\mathcal{G})) \cup \mathcal{P}_S|)$. Consequently, since by Lemma 3.2(2) $|\mathcal{P}_S| = O(|(S \cap \mathcal{V}(\mathcal{G}))|)$, $|(S \cap \mathcal{V}(\mathcal{G})) \cup U_3 \cup U_1| = O(|S|)$ as desired.

To bound the number of edges in $\mathcal{G}\diamondsuit S$, we first examine the multiple edges. Let u and v be adjacent vertices in $\mathcal{G}\diamondsuit S$. Let $X_{u,v}$ be the set of faces in $(S\cap\mathcal{F}(\mathcal{G}))\cup\mathcal{P}_S$ whose boundaries contain both u and v. Then $|X_{u,v}| \geq 1$. By Lemma 3.2(1), $|X_{u,v}| \leq 2$. If $X_{u,v} = \{F\}$, then the two boundary paths of F between u and v may degenerate into at most two multiple edges between u and v in $\mathcal{G}\diamondsuit S$. If $X_{u,v} = \{F_1, F_2\}$, then by the triconnectivity of $\mathcal{G}$, F_1 and F_2 share exactly one common boundary edge e, which is also an edge in $\mathcal{G}\diamondsuit S$. Let C_i be the boundary of F_i without e. C_1 and C_2 may degenerate into at most two multiple edges between u and v in $\mathcal{G}\diamondsuit S$. In summary, there are at most three multiple edges between two vertices in $\mathcal{G}\diamondsuit S$. Similarly, only the boundary of a face in $S \cap \mathcal{F}(\mathcal{G})$ can degenerate into a self-loop in $\mathcal{G}\diamondsuit S$; so $\mathcal{G}\diamondsuit S$ has only $O(|S|)$ self-loops. By Euler's formula, $\mathcal{G}\diamondsuit S$ has $O(|S|)$ edges as desired. ∎

3.3. Algorithms for the ACF problem. Throughout this subsection, let $\mathcal{D}_1, \ldots, \mathcal{D}_q$ be the vf-families in $\mathcal{N}$. To solve the ACF problem recursively, we use simplification to reduce the number of $\mathcal{D}_i$ and the number of sets in each $\mathcal{D}_i$.

For brevity, we define several notations. For a vf-family $\mathcal{D}$ of $\mathcal{H}$, let $\mathcal{H}\diamondsuit\mathcal{D} = \mathcal{H}\diamondsuit(\cup_{S\in\mathcal{D}}S)$. For a vf-sequence $\mathcal{N}'$: $\mathcal{D}'_1, \ldots, \mathcal{D}'_p$ of $\mathcal{H}$, let $\mathcal{H}\diamondsuit\mathcal{N}' = \mathcal{H}\diamondsuit(\mathcal{D}'_1\cup\cdots\cup\mathcal{D}'_p)$. For a vf-set S^* of $\mathcal{H}$ and a vf-family $\mathcal{D}$ of $\mathcal{H}$, we say $\mathcal{D} \leq S^*$ if $S \subseteq S^*$ for all $S \in \mathcal{D}$. For a vf-set S^* of $\mathcal{H}$, we say $\mathcal{N} \leq S^*$ if $\mathcal{D}_i \leq S^*$ for all $\mathcal{D}_i$, $1 \leq i \leq q$.

Lemmas 3.5 and 3.6 below reduce to 1 the number of $\mathcal{D}_i$ in $\mathcal{N}$ in the ACF problem.

LEMMA 3.5. *Assume $q \geq 2$. Let $\mathcal{N}_\ell = \mathcal{D}_1, \ldots, \mathcal{D}_{\lceil q/2\rceil}$ and $\mathcal{N}_r = \mathcal{D}_{\lceil q/2\rceil+1}, \ldots, \mathcal{D}_q$. Let $\mathcal{H}_\ell = \mathcal{H}\diamondsuit\mathcal{N}_\ell$ and $\mathcal{H}_r = \mathcal{H}\diamondsuit\mathcal{N}_r$.*

1. *Given $\mathcal{H}$ and $\mathcal{N}$, we can compute $\mathcal{H}_\ell$ and $\mathcal{H}_r$ in $O(|\mathcal{H}| + |\mathcal{N}|)$ total time.*
2. *For $1 \leq i \leq \lceil q/2\rceil$, $\mathcal{H}\diamondsuit\mathcal{D}_i = \mathcal{H}_\ell\diamondsuit\mathcal{D}_i$. Similarly, for $\lceil q/2\rceil + 1 \leq i \leq q$, $\mathcal{H}\diamondsuit\mathcal{D}_i = \mathcal{H}_r\diamondsuit\mathcal{D}_i$.*
3. *If $\mathcal{H}$ simplifies $\mathcal{G}$ over a vf-set S^* with $\mathcal{N} \leq S^*$, then $|\mathcal{H}_\ell| = O(|\mathcal{N}_\ell|)$ and $|\mathcal{H}_r| = O(|\mathcal{N}_r|)$.*

Proof. The three statements follow from those of Lemma 3.4, respectively. ∎

LEMMA 3.6. *Assume $q \geq 1$. Let $\mathcal{H}_i = \mathcal{H}\diamondsuit\mathcal{D}_i$.*

1. *$\mathrm{ACF}(\mathcal{H}, \mathcal{D}_i) = \mathrm{ACF}(\mathcal{H}_i, \mathcal{D}_i)$.*
2. *If $\mathcal{H}$ simplifies $\mathcal{G}$ over a vf-set S^* with $\mathcal{N} \leq S^*$, then $|\mathcal{H}_i| = O(|\mathcal{D}_i|)$.*
3. *If $\mathcal{H}$ simplifies $\mathcal{G}$ over a vf-set S^* with $\mathcal{N} \leq S^*$, then, given $\mathcal{H}$ and $\mathcal{N}$, we can compute all $\mathcal{H}_i$ in $O(|\mathcal{H}| + |\mathcal{N}|\log(q+1))$ total time.*

Proof. We prove the statements separately as follows.

Statement 1. The proof is straightforward. Note that a positive face in $\mathcal{H}_i$ is also a positive face in $\mathcal{H}$ and that a negative face in $\mathcal{H}_i$ combines one or more faces not in $\mathrm{ACF}(\mathcal{H}, \mathcal{D}_i)$.

Statement 2. The proof follows from Lemma 3.4(3).

Statement 3. The graphs $\mathcal{H}_i$ can be computed by applying Lemma 3.5 recursively with $O(\log(q+1))$ iterations. By Lemma 3.5(1), the first iteration takes $O(|\mathcal{H}|+|\mathcal{N}|)$ time. By Lemmas 3.5(3) and 3.5(1), each subsequent iteration takes $O(|\mathcal{N}|)$ time. By Lemma 3.4(2), the constant coefficient in the $O(|\mathcal{N}|)$ term does not accumulate over recursions. ∎

Lemma 3.7 below solves the ACF problem with only one $\mathcal{D}_i$ in $\mathcal{N}$.

LEMMA 3.7. *Let $\mathcal{D} = \{S_1, \ldots, S_d\}$ be a vf-family of $\mathcal{H}$ where $d \geq 1$. Let $\mathcal{D}'_\ell = \{S_1, \ldots, S_{\lceil d/2 \rceil}\}$ and $\mathcal{D}'_r = \{S_{\lceil d/2 \rceil + 1}, \ldots, S_d\}$. Let $\mathcal{H}_\ell = \mathcal{H} \diamondsuit \mathcal{D}'_\ell$ and $\mathcal{H}_r = \mathcal{H} \diamondsuit \mathcal{D}'_r$. Let $\mathcal{D}'' = \{\mathrm{ACF}(\mathcal{H}_\ell, \mathcal{D}'_\ell), \mathrm{ACF}(\mathcal{H}_r, \mathcal{D}'_r)\}$.*

1. *$\mathrm{ACF}(\mathcal{H}, \mathcal{D}) = \mathrm{ACF}(\mathcal{H}, \mathcal{D}'')$.*
2. *If $\mathcal{H}$ simplifies $\mathcal{G}$ over a vf-set S^* with $\mathcal{D} \leq S^*$, then, given $\mathcal{H}$ and $\mathcal{D}$, $\mathrm{ACF}(\mathcal{H}, \mathcal{D})$ can be computed in $O(|\mathcal{H}| + |\mathcal{D}| \log(d+1))$ time.*

Proof. The statements are proved separately as follows.

Statement 1. Note that $\mathrm{ACF}(\mathcal{H}, \mathcal{D}) = \mathrm{ACF}(\mathcal{H}, \{\mathrm{ACF}(\mathcal{H}, \mathcal{D}'_\ell), \mathrm{ACF}(\mathcal{H}, \mathcal{D}'_r)\})$ by a straightforward case analysis. Then, as in Lemma 3.6(1), $\mathrm{ACF}(\mathcal{H}, \mathcal{D}'_\ell) = \mathrm{ACF}(\mathcal{H}_\ell, \mathcal{D}'_\ell)$ and $\mathrm{ACF}(\mathcal{H}, \mathcal{D}'_r) = \mathrm{ACF}(\mathcal{H}_r, \mathcal{D}'_r)$.

Statement 2. We compute $\mathrm{ACF}(\mathcal{H}, \mathcal{D})$ recursively using statement 1. If $d = 1$, then $\mathrm{ACF}(\mathcal{H}, \mathcal{D}) = S_1$. If $d = 2$, then $\mathrm{ACF}(\mathcal{H}, \mathcal{D})$ can be computed in $O(|\mathcal{H}|)$ time in a straightforward manner. For $d > 2$, there are three stages:

1. Compute $\mathcal{H}_\ell$ and $\mathcal{H}_r$ in $O(|\mathcal{H}| + |D|)$ time in a straightforward manner.
2. Recursively compute $\mathrm{ACF}(\mathcal{H}_\ell, \mathcal{D}'_\ell)$ and $\mathrm{ACF}(\mathcal{H}_r, \mathcal{D}'_r)$.
3. Compute $\mathrm{ACF}(\mathcal{H}, \mathcal{D}'')$ in $O(|\mathcal{H}|)$ time in a straightforward manner, which is $\mathrm{ACF}(\mathcal{H}, \mathcal{D})$ by statement 1.

This recursive computation has $\log d + O(1)$ iterations. The recursion at the top level takes $O(|\mathcal{H}| + |\mathcal{D}|)$ time. Every subsequent level takes $O(|\mathcal{D}|)$ time since by Lemma 3.4(3) $O(|\mathcal{H}_\ell|) = O(|\mathcal{D}'_\ell|)$ and $O(|\mathcal{H}_r|) = O(|\mathcal{D}'_r|)$. Note that by Lemma 3.4(2), the constant coefficient in the $O(|\mathcal{D}|)$ term does not accumulate over recursions. ∎

The next theorem is the main result of this section.

THEOREM 3.8.

1. *Let d be the maximum number of vf-sets in any $\mathcal{D}_i$ in $\mathcal{N}$. If $\mathcal{H}$ simplifies $\mathcal{G}$ over a vf-set S^* with $\mathcal{N} \leq S^*$, then the ACF problem can be solved in $O(|\mathcal{H}| + |\mathcal{N}| \log(d+q))$ time.*
2. *Let d be the maximum number of vertex sets in any $\mathcal{C}_i$ in $\mathcal{M}$. Case* M3 *of the CFE problem can be solved in $O(|\mathcal{G}| + |\mathcal{M}| \log(d+q))$ time.*

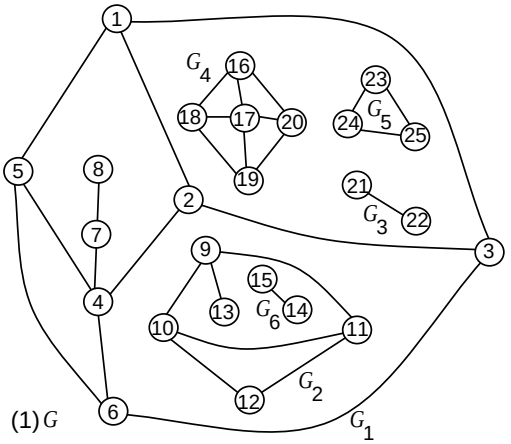
Proof. Statement 1 follows from Lemmas 3.6 and 3.7. Statement 2 follows from Observation 3.1, statement 1, and the fact that $\mathcal{G}$ has a unique combinatorial embedding computable in linear time [21, 30]. ∎

In section 6.4, the algorithm for Case M2 of the CFE problem calls Theorem 3.8(2) to solve subproblems in which some $S \in \mathcal{C}_i$ may consist of a single edge $\{u, v\}$. For such subproblems, we replace S by $\{u\}$ and $\{v\}$ and then apply Theorem 3.8(2).

4. Reducing Theorem 2.2 to Case M1, where $\mathcal{G}$ is connected. Let $\mathcal{G}_1, \ldots, \mathcal{G}_k$ be the connected components of $\mathcal{G}$. Let $\mathcal{C}_1, \ldots, \mathcal{C}_q$ be the families in $\mathcal{M}$. A family $\mathcal{C}_h$ in $\mathcal{M}$ is *global* if, for every $i \in \{1, \ldots, k\}$, $\mathcal{C}_h$ is not $\mathcal{G}_i$-local. Let H be an edge-labeled graph defined as follows. The vertices of H are $1, \ldots, k$. For each global $\mathcal{C}_h$, H contains a cycle C possibly of length 2, where (1) the vertices of C are those $i \in \{1, \ldots, k\}$ such that some set in $\mathcal{C}_h$ is $\mathcal{G}_i$-local and (2) the edges of C are all labeled h. See Figures 4.1(1) through 4.1(3) for examples of $\mathcal{G}$, $\mathcal{M}$, and H.

OBSERVATION 4.1. *Let $H_1, \ldots, H_\ell$ be the connected components of H. For each H_j, let $\mathcal{G}'_j$ be the subgraph of $\mathcal{G}$ formed by all $\mathcal{G}_i$ with $i \in \mathcal{V}(H_j)$. Let $\mathcal{M}'_j$ be the sequence of all $\mathcal{G}'_j$-local families in $\mathcal{M}$. Then $\mathcal{G}$ satisfies $\mathcal{M}$ if and only if every $\mathcal{G}'_j$ satisfies $\mathcal{M}'_j$.*

By Observation 4.1, we may assume that H is connected. Let $B_1, \ldots, B_p$ be the blocks of H. Then, for each global $\mathcal{C}_h$, exactly one B_j contains all the edges labeled h. For every B_j, let $\mathcal{U}_j = \cup_h \mathcal{C}_h$, where h ranges over all labels on the edges of B_j. For



C_1 = { {2,4,6}, {17,18}, {21} }

C_2 = { {1,5}, {23,24}, {17,20} }

C_3 = { {4,5}, {9,13} }

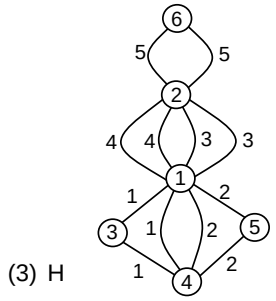
C_4 = { {2,3}, {9,11} }

C_5 = { {13}, {14,15} }

C_6 = { {1,3}, {4,6} }

C_7 = { {1,2}, {7,8} }

(2) M



M_1 : { {1,3}, {4,6} }, { {1,2}, {7,8} }, { {2,3}, {4,5} }, { {2,4,6}, {1,5} }

M_2 : { {9,13}, {9,11} }, { {13} }

M_3 : { {21} }

M_4 : { {17,18}, {17, 20} }

M_5 : { {23,24} }

M_6 : { {14,15} }

(4)

FIG. 4.1. (1) *This is a simple disconnected graph $\mathcal{G}$ with six connected components $\mathcal{G}_1$ through $\mathcal{G}_6$, where $\mathcal{V}(\mathcal{G}_1) = \{1, \ldots, 8\}$, $\mathcal{V}(\mathcal{G}_2) = \{9, \ldots, 13\}$, $\mathcal{V}(\mathcal{G}_3) = \{21, 22\}$, $\mathcal{V}(\mathcal{G}_4) = \{16, \ldots, 20\}$, $\mathcal{V}(\mathcal{G}_5) = \{23, \ldots, 25\}$, and $\mathcal{V}(\mathcal{G}_6) = \{14, 15\}$.* (2) *This is a sequence $\mathcal{M}$ of families of vertex subsets of $\mathcal{G}$, where $\mathcal{C}_6$ and $\mathcal{C}_7$ are $\mathcal{G}_1$-local but the rest of the families are global.* (3) *This is the graph H constructed from $\mathcal{G}$ and $\mathcal{M}$.* (4) *These are the sequences constructed for $\mathcal{G}_1$ through $\mathcal{G}_6$.*

each $\mathcal{G}_i$, let $\mathcal{M}_i$ be the sequence consisting of the $\mathcal{G}_i$-local families in $\mathcal{M}$ as well as the families $\mathcal{U}_{j,i} = \{U \in \mathcal{U}_j \mid U \text{ is } \mathcal{G}_i\text{-local}\}$ for all B_j with $i \in \mathcal{V}(B_j)$. See Figure 4.1(4) for an example of $\mathcal{M}_1, \ldots, \mathcal{M}_6$ constructed from $\mathcal{G}$, $\mathcal{M}$, and H in Figures 4.1(1) through 4.1(3).

LEMMA 4.2. *$\mathcal{G}$ satisfies $\mathcal{M}$ if and only if every $\mathcal{G}_i$ satisfies $\mathcal{M}_i$.*

Proof. The two directions are proved as follows.

($\Longrightarrow$) Let Φ be an embedding of $\mathcal{G}$ satisfying $\mathcal{M}$. Let Φ_i be the restriction of Φ to $\mathcal{G}_i$. For each $\mathcal{G}_i$, our goal is to prove that Φ_i satisfies $\mathcal{M}_i$. First, Φ_i satisfies each $\mathcal{G}_i$-local family in $\mathcal{M}$. Let B_j be a block of H with $i \in B_j$. We next prove that Φ_i satisfies $\mathcal{U}_{j,i}$. Let $i, i_1, \ldots, i_\ell$ be the vertices of B_j. We claim that $\mathcal{G}$ has no cycle C such that at least one but not all of $\mathcal{G}_i$, $\mathcal{G}_{i_1}, \ldots, \mathcal{G}_{i_\ell}$ are inside C in Φ. To prove by contradiction, assume that such a C exists. Then some $\mathcal{G}_x$ with $1 \le x \le k$ contains C. However, by the construction of H, no connected component of $H - \{x\}$ contains all of i, $i_1, \ldots, i_\ell$, contradicting the fact that B_j is a block of H. Thus the claim holds. Therefore, the boundary of some face F in Φ intersects each of $\mathcal{G}_i, \mathcal{G}_{i_1}, \ldots, \mathcal{G}_{i_\ell}$. Since F must be unique, the boundary of F intersects every set in $\mathcal{C}_h$ for every $\mathcal{C}_h$ in $\mathcal{M}$ such that the sets in $\mathcal{C}_h$ fall into two or more of $\mathcal{G}_i$, $\mathcal{G}_{i_1}$, ..., $\mathcal{G}_{i_\ell}$. Hence the boundary of F intersects every set in $\mathcal{U}_j$. Consequently, Φ_i satisfies $\mathcal{U}_{j,i}$.

($\Longleftarrow$) Let Φ_i be an embedding of $\mathcal{G}_i$ satisfying $\mathcal{M}_i$. We construct an embedding

of $\mathcal{G}$ satisfying $\mathcal{M}$ as follows. First, consider a block B_j of H. Let $i_1, \ldots, i_\ell$ be the vertices of B_j. Let $\mathcal{G}'_j$ be the subgraph of $\mathcal{G}$ formed by $\mathcal{G}_{i_1}, \ldots, \mathcal{G}_{i_\ell}$. Let $\mathcal{M}'_j$ be the sequence consisting of $\mathcal{U}_j$ and the $\mathcal{G}_{i_x}$-local families in $\mathcal{M}$ for $x = 1, \ldots, \ell$. We can assume that the boundary of the exterior face of Φ_{i_x} intersects every set in $\mathcal{U}_{j,i_x}$. By identifying the exterior faces of $\Phi_{i_1}, \ldots, \Phi_{i_\ell}$, we can combine the embeddings into an embedding Φ'_j of $\mathcal{G}'_j$ satisfying $\mathcal{M}'_j$. Next, we utilize $T = \Psi(H)$ to combine $\Phi'_1, \ldots, \Phi'_p$ into a single embedding of $\mathcal{G}$. First, root T at a block of H. For a leaf B_{j_1} in T, let $\mathcal{G}_i$ and B_{j_2} be the parent and grandparent of B_{j_1} in T, respectively. Let $\mathcal{L}_{i,1}$ (respectively, $\mathcal{L}_{i,2}$) be the restriction of Φ'_{j_1} (respectively, Φ'_{j_2}) to $\mathcal{G}_i$. Note that Φ_i, $\mathcal{L}_{i,1}$, and $\mathcal{L}_{i,2}$ are topologically equivalent up to the choice of their exterior face. Thus Φ'_{j_1} (respectively, Φ'_{j_2}) can be obtained as follows: For every vertex $i' \neq i$ of B_{j_1} (respectively, B_{j_2}), put a suitable embedding $\mathcal{L}_{i'}$ of $\mathcal{G}_{i'}$ that is topologically equivalent to $\Phi_{i'}$ into a suitable face $F_{i'}$ of Φ_i. This gives an embedding of those $\mathcal{G}_x \in \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$ with $x \in \mathcal{V}(B_{j_1}) \cup \mathcal{V}(B_{j_2})$. We replace Φ'_{j_2} with this embedding, replace B_{j_2} with the union of B_{j_1} and B_{j_2}, and delete B_{j_1} from T. Afterward, if $\mathcal{G}_i$ becomes a leaf of T, then we further delete it from T. We repeat this process until T is a single vertex, at which time we obtain an embedding of $\mathcal{G}$ satisfying $\mathcal{M}$. □

THEOREM 4.3. *Theorem* 2.2 *holds if it holds for Case* M1.

Proof. The proof follows from Lemma 4.2 and the fact that H and the sequences $\mathcal{M}_i$ above can be constructed from $\mathcal{G}$ and $\mathcal{M}$ in $O(I)$ time. □

5. Reducing Case M1 to Case M2, where $\mathcal{G}$ is biconnected. This section assumes Case M1, where $\mathcal{G}$ is connected. We also assume that $\mathcal{G}$ has at least two vertices; otherwise, the problem is trivial.

Section 5.1 shows how to eliminate one cut vertex from $\mathcal{G}$; iterating this elimination until $\mathcal{G}$ has no cut vertex gives us a reduction from Case M1 to Case M2. However, this reduction is not efficient. Section 5.2 describes a more efficient reduction based on a direct elimination of all cut vertices from $\mathcal{G}$. Throughout the rest of this section, let $\mathcal{C}_1, \ldots, \mathcal{C}_q$ be the families in $\mathcal{M}$.

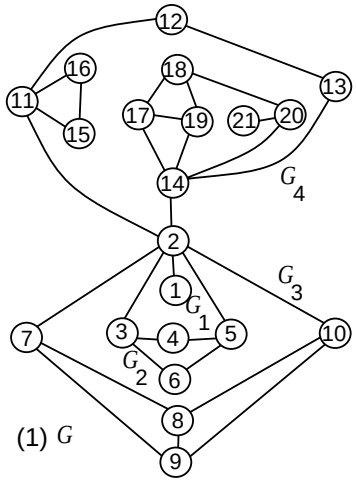
5.1. Eliminating one cut vertex. Let w be a cut vertex of $\mathcal{G}$. Let $\mathcal{G}_1, \ldots, \mathcal{G}_k$ be the augmented components of $\mathcal{G}$ induced by w. For each $\mathcal{C}_h$ in $\mathcal{M}$, let $U_{h,1}, \ldots, U_{h,t_h}$ be the sets in $\mathcal{C}_h$ containing w; possibly $t_h = 0$. $\mathcal{C}_h$ is *w-global* if, for all $i \in \{1, \ldots, k\}$, $\mathcal{C}_h - \{U_{h,1}, \ldots, U_{h,t_h}\}$ is not $\mathcal{G}_i$-local; otherwise, $\mathcal{C}_h$ is *w-local*.

OBSERVATION 5.1.

1. *Assume that $\mathcal{C}_h - \{U_{h,1}, \ldots, U_{h,t_h}\}$ is $\mathcal{G}_i$-local for some $\mathcal{G}_i$. Then $\mathcal{G}$ satisfies $\mathcal{M}$ if and only if $\mathcal{G}$ satisfies $\mathcal{M}$ with $\mathcal{C}_h$ replaced by $(\mathcal{C}_h - \{U_{h,1}, \ldots, U_{h,t_h}\}) \cup \{U_{h,1} \cap \mathcal{V}(\mathcal{G}_i), \ldots, U_{h,t_h} \cap \mathcal{V}(\mathcal{G}_i)\}$.*
2. *Assume that $\mathcal{C}_h$ is w-global. Then $\mathcal{G}$ satisfies $\mathcal{M}$ if and only if $\mathcal{G}$ satisfies $\mathcal{M}$ with $\mathcal{C}_h$ replaced by $\mathcal{C}_h - \{U_{h,1}, \ldots, U_{h,t_h}\}$.*

By Observation 5.1, we may assume that (1) each set in a w-global family in $\mathcal{M}$ does not contain w and (2) each set in a family in $\mathcal{M}$ is $\mathcal{G}_i$-local for some $\mathcal{G}_i$. Let H be an edge-labeled graph constructed as follows. The vertices of H are $1, \ldots, k$. For each w-global family $\mathcal{C}_h$, H has a cycle C possibly of length 2, where (1) the vertices of C are those $i \in \{1, \ldots, k\}$ such that at least one set in $\mathcal{C}_h$ is $\mathcal{G}_i$-local and (2) the edges of C are all labeled h. See Figures 5.1(1) through 5.1(3) for examples of $\mathcal{G}$, $\mathcal{M}$, and H.

Note that Observation 4.1 still holds for this H and the augmented components $\mathcal{G}_1, \ldots, \mathcal{G}_k$. Thus we may assume that H is connected. Let $B_1, \ldots, B_p$ be the blocks of H. Clearly, for each w-global family $\mathcal{C}_h \in \mathcal{M}$, exactly one block of H contains all the edges labeled h. For each B_j, let $\mathcal{U}_j = \cup_h \mathcal{C}_h \cup \{\{w\}\}$, where h ranges over



C_1 = { {2,7,8,11}, {11,16}, {17,19} }

C_2 = { {2,14,17}, {1}, {4} }

C_3 = { {6}, {8,9} }

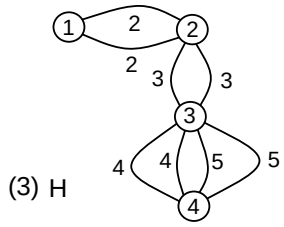
C_4 = { {9,10}, {13,14}, {11,15} }

C_5 = { {7,9}, {12,13} }

C_6 = { {14, 20, 21}, {15,16} }

C_7 = { {19}, {21} }

(2) M



M_1 : { {2}, {1} }

M_2 : { {2}, {4} }, { {2}, {6} }

M_3 : { {2}, {8,9} }, { {2}, {9,10}, {7,9} }

M_4: { {2,11}, {11,16}, {17,19} }, { {14, 20, 21}, {15,16} }, { {2}, {13,14}, {11,15}, {12,13} }, { {19}, {21} }

(4)

FIG. 5.1. (1) *This is a simple connected graph $\mathcal{G}$ with a cut vertex* 2. *It induces four augmented components $\mathcal{G}_1$ through $\mathcal{G}_4$ with $\mathcal{V}(\mathcal{G}_1) = \{1, 2\}$, $\mathcal{V}(\mathcal{G}_2) = \{2, \ldots, 6\}$, $\mathcal{V}(\mathcal{G}_3) = \{2, 7, \ldots, 10\}$, and $\mathcal{V}(\mathcal{G}_4) = \{2, 11, \ldots, 21\}$.* (2) *This is a sequence $\mathcal{M}$ of families of vertex subsets of $\mathcal{G}$, where only $\mathcal{C}_2$ through $\mathcal{C}_5$ are* 2*-global.* (3) *This is the graph H constructed from $\mathcal{G}$ and $\mathcal{M}$.* (4) *These are the sequences constructed for $\mathcal{G}_1$ through $\mathcal{G}_4$, respectively.*

all labels on the edges of B_j. For each $\mathcal{G}_i$, let $\mathcal{M}_i$ be the sequence consisting of the $\mathcal{G}_i$-local families in $\mathcal{M}$ as well as the families $\mathcal{U}_{j,i} = \{U \in \mathcal{U}_j \mid U \text{ is } \mathcal{G}_i\text{-local}\}$ for all B_j with $i \in \mathcal{V}(B_j)$. See Figure 5.1(4) for an example of $\mathcal{M}_1, \ldots, \mathcal{M}_4$ constructed from $\mathcal{G}$, $\mathcal{M}$, and H in Figures 5.1(1) through 5.1(3).

LEMMA 5.2. *$\mathcal{G}$ satisfies $\mathcal{M}$ if and only if every $\mathcal{G}_i$ satisfies $\mathcal{M}_i$.*

Proof. The two directions are proved as follows.

($\Longrightarrow$) The proof is the same as that of Lemma 4.2 except that the claim therein now implies that the boundary of some face F in Φ intersects each of $\mathcal{G}_i - \{w\}, \mathcal{G}_{i_1} - \{w\}, \ldots, \mathcal{G}_{i_\ell} - \{w\}$.

($\Longleftarrow$) The proof is the same as that of Lemma 4.2 except that Φ'_{j_1} (respectively, Φ'_{j_2}) now can be obtained as follows: For each vertex $i' \neq i$ of B_{j_1} (respectively, B_{j_2}), put a suitable embedding $\mathcal{L}_{i'}$ of $\mathcal{G}_{i'}$ that is topologically equivalent to $\Phi_{i'}$ into a suitable face $F_{i'}$ of Φ_i, and then identify the two occurrences of w. □

5.2. Eliminating all cut vertices. Let $T = \Psi(\mathcal{G})$. Root T at a block vertex, and perform a postorder traversal of T. For each vertex γ of T, let $\text{post}(\gamma)$ be the postorder number of γ in the postorder traversal of T.

Let $W = \{w_1, \ldots, w_\ell\}$ be the set of cut vertices of $\mathcal{G}$ where $\text{post}(w_1) < \cdots < \text{post}(w_\ell)$. For each $v \in \mathcal{V}(\mathcal{G}) - W$, let $\text{post}(v) = \text{post}(B)$, where B is the unique block of $\mathcal{G}$ with $v \in \mathcal{V}(B)$. We may assume $\mathcal{V}(\mathcal{G}) = \{1, \ldots, n\}$. For each $v \in \mathcal{V}(\mathcal{G})$, the *rank*

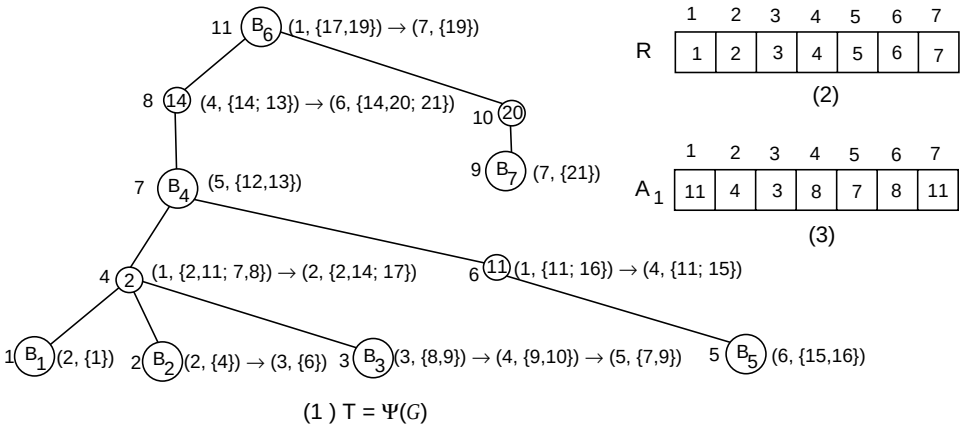


FIG. 5.2. (1) *This is* $\Psi(\mathcal{G})$, *where* $\mathcal{G}$ *is the simple graph in Figure* 5.1(1). *Here,* $\mathcal{V}(B_1) = \{1, 2\}$, $\mathcal{V}(B_2) = \{2, \ldots, 6\}$, $\mathcal{V}(B_3) = \{2, 7, \ldots, 10\}$, $\mathcal{V}(B_4) = \{2, 11, \ldots, 14\}$, $\mathcal{V}(B_5) = \{11, 15, 16\}$, $\mathcal{V}(B_6) = \{14, 17, \ldots, 20\}$, *and* $\mathcal{V}(B_7) = \{20, 21\}$. *The number to the left of each vertex* γ *of* $\Psi(\mathcal{G})$ *is* $\text{post}(\gamma)$, *and the list to the right is* $L(\gamma)$ *before processing the first cut vertex of* $\mathcal{G}$. *For visibility, each set* U *in a pair in* $L(\gamma)$ *with* $U \cap W \neq \emptyset$ *is divided into two parts via a semicolon; the first part consists of vertices in* $U \cap W$ *in the increasing order of their postorder numbers.* (2) *These are the representatives in the union-find data structure before processing the first cut vertex of* $\mathcal{G}$. (3) *This is the array* A_1 *before processing the first cut vertex of* $\mathcal{G}$.

of v, denoted by $\text{rank}(v)$, is $(\text{post}(v), v)$. The rank of a vertex u is *lower* than that of another vertex v if (1) $\text{post}(u) < \text{post}(v)$ or (2) $\text{post}(u) = \text{post}(v)$ and $u < v$. For each $w_i \in W$, let $B_{i,1}, \ldots, B_{i,k_i}$ be the children of w_i in T. Let $B_{i,0}$ be the parent of vertex w_i in T.

THEOREM 5.3. *Theorem* 2.2 *holds for Case* M1 *if it holds for Case* M2.

Proof. It suffices to construct a sequence $\mathcal{M}[B]$ for each block B of $\mathcal{G}$, with a total size of $O(I)$ in $O(I \log I)$ total time over all the blocks of $\mathcal{G}$, such that $\mathcal{G}$ satisfies $\mathcal{M}$ if and only if every B satisfies $\mathcal{M}[B]$. To construct $\mathcal{M}[B]$ based on Observation 5.1 and Lemma 5.2, we process $w_1, \ldots, w_\ell$ one at a time. During the processing of w_i, we construct $\mathcal{M}[B_{i,j}]$ for all $j = 1, \ldots, k_i$. Then, we delete w_i, $B_{i,1}, \ldots, B_{i,k_i}$ from T. After processing w_ℓ, we are left with the root $B_{\ell,0}$ for which we then construct $\mathcal{M}[B_{\ell,0}]$.

We use the following data structures. See Figure 5.2 for an example of some of the data structures before processing the first cut vertex of $\mathcal{G}$.

1. During the construction, some families in $\mathcal{M}$ may be united, and we use a union-find data structure to maintain a collection of disjoint dynamic subsets of $\Delta = \{1, \ldots, q\}$. (Recall that q is the number of families in $\mathcal{M}$.) Each subset of Δ in the data structure is identified by a *representative* member of the subset. For each $h \in \Delta$, let $R(h)$ be the representative of the subset containing h. Initially, each $h \in \Delta$ forms a singleton subset, and thus $R(h) = h$.
2. Each set U in a family in $\mathcal{M}$ is implemented as a pair $(\mathcal{W}[U], \mathcal{S}[U])$, where $\mathcal{W}[U]$ is a linked list, and $\mathcal{S}[U]$ is a splay tree [28]. Initially, $\mathcal{W}[U]$ consists of the vertices in $U \cap W$ in the increasing order of their postorder numbers. $\mathcal{S}[U]$ is initialized by inserting the ranks of the vertices in $U - W$ into an empty splay tree. A splay tree supports the following operations in amortized logarithmic time per operation: (1) insert a rank, and (2) delete the ranks in a given range.

3. A linked list $L[B]$ for each block B of $\mathcal{G}$. Initially, each $L[B]$ consists of all pairs (h, U) such that $h \in \Delta$, $U \in \mathcal{C}_h$, U is B-local, and $U \cap W = \emptyset$.
4. A linked list $L[w_i]$ for each $w_i \in W$. Initially, each $L[w_i]$ consists of all pairs (h, U) such that $h \in \Delta$, $U \in \mathcal{C}_h$, $w_i \in U$, and $i = \min\{j \mid w_j \in U \cap W\}$.
5. An array $A_1[1 \ldots q]$ of integers. Initially, for each $h \in \Delta$, $A_1[h] = \max_\gamma \text{post}(\gamma)$, where γ ranges over all vertices of T such that $L[\gamma]$ contains a pair $(h, *)$ with $* =$ "don't care."
6. An array $A_2[1 \ldots q]$ of integers. Initially, for each $h \in \Delta$, $A_2[h] = 0$.
7. An array $J[1 \ldots q]$ of linked lists of integers. Initially, for each $h \in \Delta$, $J[h]$ is empty.
8. A temporary array $Y[1 \ldots q]$ of integers.

We maintain the following invariants immediately before processing each w_i. In particular, we initialize the above data structures so that the invariants hold before w_1 is processed. It takes $O(I)$ total time to initialize the data structures except the splay trees.

1. For each vertex γ of T and each pair $(h, U) \in L[\gamma]$, (1) $\mathcal{W}[U]$ consists of the vertices in $U \cap \{w_i, \ldots, w_\ell\}$ in the increasing order of their postorder numbers, (2) the rank of each vertex of $U - \{w_i, \ldots, w_\ell\}$ is stored in $\mathcal{S}[U]$, and (3) for every $w_j \in U \cap \{w_1, \ldots, w_{i-1}\}$, $\text{post}(w_j)$ and $\text{rank}(w_j)$ have been updated as $\text{post}(B_{j,0})$ and $(\text{post}(B_{j,0}), w_j)$, respectively.
2. For each block vertex B of T and each $(h, U) \in L[B]$, it holds that $h \in \Delta$, U is B-local, and $U \cap \{w_i, \ldots, w_\ell\} = \emptyset$.
3. For each $j \in \{i, \ldots, \ell\}$ and each $(h, U) \in L[w_j]$, it holds that $h \in \Delta$, $w_j \in U$, and $j = \min\{x \mid i \le x \le \ell \text{ and } w_x \in U\}$.
4. For each $h \in \Delta$ with $R(h) = h$, let $\mathcal{C}'_h = \{U \mid \text{there is a vertex } \gamma \text{ of } T \text{ such that } L[\gamma] \text{ contains a pair } (h', U) \text{ with } R(h') = h\}$. Let $\mathcal{M}'$ be the sequence of all families $\mathcal{C}'_h$ such that $h \in \Delta$ and $R(h) = h$. Let $\mathcal{G}'$ be the subgraph of $\mathcal{G}$ induced by $\cup_B \mathcal{V}(B)$, where B ranges over all the block vertices of T. Then $\mathcal{G}$ satisfies $\mathcal{M}$ if and only if (1) $\mathcal{G}'$ satisfies $\mathcal{M}'$ and (2) for each block B of $\mathcal{G}$ that has been deleted from T, B satisfies $\mathcal{M}[B]$.
5. For each $h \in \Delta$ with $R(h) = h$, $A_1[h] = \max_\gamma \text{post}(\gamma)$, where γ ranges over all vertices of T such that $L[\gamma]$ contains a pair $(h', *)$ with $R(h') = h$.
6. For each $h \in \Delta$, $A_2[h] = 0$ and $J[h]$ is empty.

We process w_i as follows in Stages W1 through W4. See Figure 5.3 for an example of some of the data structures after processing the first cut vertex of $\mathcal{G}$.

• Stage W1 checks whether each related family is w_i-global as follows.

1. Compute $X = \{h \in \Delta \mid R(h) = h, \text{ and for some } j \in \{1, \ldots, k_i\}, L[B_{i,j}] \text{ contains a pair } (h', *) \text{ with } R(h') = h\}$. (*Remark.* For each $h \in \Delta - X$ with $R(h) = h$, the family $\mathcal{C}'_h - \{U \mid w_i \in U\}$ is Q_i-local, where Q_i is the augmented component of $\mathcal{G}'$ induced by w_i that is not among $B_{i,1}, \ldots, B_{i,k_i}$. See the fourth invariant for $\mathcal{C}'_h$ and $\mathcal{G}'$.)
2. For each $h \in X$, set $Y[h]$ to be the number of integers $j \in \{1, \ldots, k_i\}$ such that $L[B_{i,j}]$ contains a pair $(h', *)$ with $R(h') = h$. (*Remark.* For $h \in X$, $Y[h] \ge 1$.)
3. For each $h \in X$, perform the following:
   (a) If $Y[h] = 1$ and $A_1[h] \le \text{post}(w_i)$, then set $A_2[h] = j$, where j is the unique integer in $\{1, \ldots, k_i\}$ such that $L[B_{i,j}]$ contains a pair $(h', *)$ with $R(h') = h$. (*Remark.* $\mathcal{C}'_h - \{U \mid w_i \in U\}$ is $B_{i,j}$-local.)
   (b) Otherwise, set $A_2[h] = -1$. (*Remark.* $\mathcal{C}'_h - \{U \mid w_i \in U\}$ is w_i-global.)

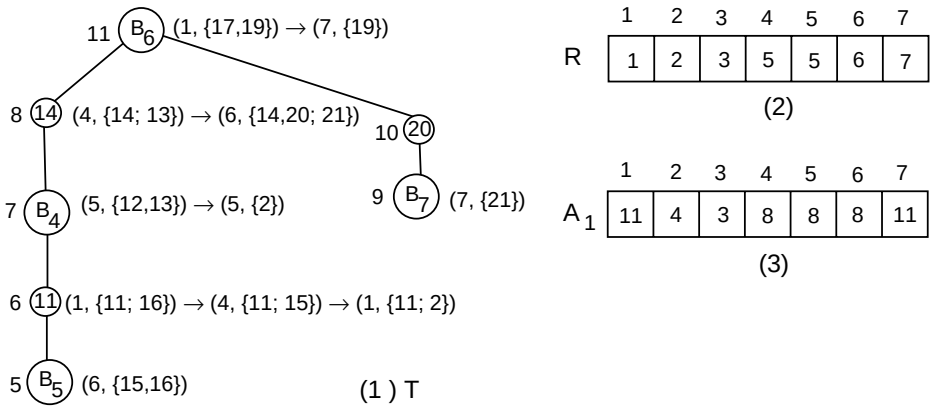


FIG. 5.3. *This is the data structure after processing the first cut vertex (i.e., the vertex* 2*) of the graph in Figure* 5.3(1).

• Stage W2 modifies each U with $w_i \in U$ in each w_i-local family based on Observation 5.1(1) as follows.

1. For each $(h, U) \in L[w_i]$ with $A_2[R(h)] \geq 1$, let $j = A_2[R(h)]$, delete all vertices outside $\mathcal{V}(B_{i,j})$ from U, and then insert (h, U) into $L[B_{i,j}]$. Here, deleting all vertices outside $\mathcal{V}(B_{i,j})$ from U is done as follows: Delete w_i from $\mathcal{W}[U]$, delete all the ranks in the range $[-\infty \ldots (\text{post}(B_{i,j}), 0)]$ and all the ranks in the range $[(\text{post}(B_{i,j}), n+1) \ldots \infty]$ from $\mathcal{S}[U]$, and insert $(\text{post}(B_{i,j}), w_i)$ into $\mathcal{S}[U]$.
2. For each $(h, U) \in L[w_i]$ with $A_2[R(h)] = 0$, perform the following. (*Remark.* $\mathcal{C}'_h - \{U \mid w_i \in U\}$ is Q_i-local. See the remark in step 1 of Stage W1 for Q_i.)
   (a) Delete all vertices v with $\text{post}(v) < \text{post}(w_i)$ from U as follows: Delete w_i from $\mathcal{W}[U]$, delete all the ranks in the range $[-\infty \ldots \text{rank}(w_i)]$ from $\mathcal{S}[U]$, and insert $(\text{post}(B_{i,0}), w_i)$ into $\mathcal{S}[U]$.
   (b) If $\mathcal{W}[U] = \emptyset$, i.e., U has no cut vertex, then insert (h, U) into $L[B_{i,0}]$, and set $A_1[R(h)] = \max\{\text{post}(B_{i,0}), A_1[R(h)]\}$.
   (c) If $\mathcal{W}[U] \neq \emptyset$, then find the first vertex w_j in $\mathcal{W}[U]$, insert (h, U) into $L[w_j]$, and set $A_1[R(h)] = \max\{\text{post}(w_j), A_1[R(h)]\}$. (*Remark.* $j > i$.)

• Stage W3 modifies each w_i-global family based on Observation 5.1(2) as follows.

1. For each $h \in X$ with $A_2[h] = -1$, set $J[h] = \{j \in \{1, \ldots, k_i\} \mid L[B_{i,j}]$ contains a pair $(h', *)$ with $R(h') = h\}$.
2. For each $h \in X$ with $A_2[h] = -1$ and $A_1[h] > \text{post}(w_i)$, insert 0 into $J[h]$.
3. Set $\text{post}(w_i) = \text{post}(B_{i,0})$ and $\text{rank}(w_i) = (\text{post}(B_{i,0}), w_i)$.
4. Construct an edge-labeled graph H_i as follows. The vertices of H_i are 0, 1, $\ldots$, k_i. For each $h \in X$ with $A_2[h] = -1$, H_i has a cycle possibly of length 2 whose vertices are the integers in $J[h]$ and whose edges are all labeled h.
5. For each block $\mathcal{B}$ of H_i, find the labels $h_1, \ldots, h_t$ on the edges in $\mathcal{B}$, and unite those subsets in the union-find data structure that have $h_1, \ldots, h_t$ as their representative, respectively; afterward, for the representative h_r of the resulting subset, further perform the following:
   (a) Insert $(h_r, \{w_i\})$ into all lists $L[B_{i,j}]$ such that $j \in \mathcal{V}(\mathcal{B})$.
   (b) If $0 \in \mathcal{V}(\mathcal{B})$, then set $A_1[h_r] = \max\{\text{post}(B_{i,0}), A_1[h_1], \ldots, A_1[h_t]\}$.

• Stage W4 constructs the sequences $\mathcal{M}[B_{i,j}]$ for $1 \leq j \leq k_i$ and updates the

data structures as follows.

1. For each j and each (h, U) in $L[B_{i,j}]$, replace (h, U) by $(R(h), U)$.
2. For each j, set $\mathcal{M}[B_{i,j}]$ to be the sequence of the families $\mathcal{C}''_h = \{U \mid (h, U) \in L[B_{i,j}]\}$, where h ranges over those integers that are in a pair in $L[B_{i,j}]$.
3. Delete w_i and its children from T.
4. For each $h \in X$, set $A_2[h] = 0$ and $J[h] = \emptyset$.

By Observation 5.1 and Lemma 5.2, after the processing of w_i, the invariants hold for $i + 1$.

After processing w_ℓ, we construct $\mathcal{M}[B_{\ell,0}]$ as follows: Replace each pair (h, U) in $L[B_{\ell,0}]$ by $(R(h), U)$, and then set $\mathcal{M}[B_{\ell,0}]$ to be the sequence of the families $\mathcal{C}''_h = \{U \mid (h, U) \in L[B_{\ell,0}]\}$, where h ranges over those integers that are in a pair in $L[B_{\ell,0}]$.

By the invariants, Observation 5.1, and Lemma 5.2, $\mathcal{G}$ satisfies $\mathcal{M}$ if and only if every block B of $\mathcal{G}$ satisfies $\mathcal{M}[B]$. As for the time complexity, we make the following observations:

1. When processing w_i, we create at most n_i new sets all equal to $\{w_i\}$, where n_i is the maximum number of blocks in a simple graph with $k_i + 1$ vertices. Since $n_i = O(k_i + 1)$ and $k_i + 1$ does not exceed the degree of w_i in $\mathcal{G}$, the total number of newly created sets is $O(|\mathcal{G}|)$.
2. If a set U does not intersect $\{w_i, \ldots, w_\ell\}$ immediately before the processing of w_i, then there is at most one $w_j \in \{w_i, \ldots, w_\ell\}$ such that some vertices of U are touched during the processing of w_j.
3. If w_i is in U immediately before the processing of w_i, then we either (1) touch at most $1 + |\{v \in U \mid \text{post}(v) \le \text{post}(w_i)\}|$ vertices of U during the processing of w_i, or (2) touch no vertex of U during the processing of each $w_j \in \{w_{i+1}, \ldots, w_\ell\}$.

There are at most q unions and $O(I)$ finds, and at most $|\mathcal{G}|$ insertions into each splay tree. By the above observations, the total time spent on the union-find data structure is $O(I \log I)$, that on the splay trees is $O(I \log |\mathcal{G}|)$, and that on the remaining computation is $O(I)$, all within the desired time. □

6. Case M2, where $\mathcal{G}$ is biconnected. This section assumes that $\mathcal{G}$ is biconnected. Let $\mathcal{C}_1, \ldots, \mathcal{C}_q$ be the families in $\mathcal{M}$. For each $i \in \{1, \ldots, q\}$, let $\mathcal{C}_i = \{U_{i,1}, \ldots, U_{i,r_i}\}$.

THEOREM 6.1. *Theorem* 2.2 *holds for Case* M2.

To prove Theorem 6.1, we review a decomposition of $\mathcal{G}$ in section 6.1, outline the basic ideas of our CFE algorithm in section 6.2, detail the algorithm in section 6.3, and analyze it in section 6.4.

6.1. SPQR decompositions. A *planar st-graph* G is a directed acyclic plane graph such that G has exactly one source s and exactly one sink t, and both vertices are on the exterior face. These two vertices are the *poles* of G. A *split pair* of G is either a pair of adjacent vertices or a pair of vertices whose removal disconnects the graph obtained from G by adding the edge (s, t). A *split component* of a split pair $\{u, v\}$ is either an edge (u, v) or a maximal subgraph C of G such that C is a planar uv-graph and $\{u, v\}$ is not a split pair of C. A split pair $\{u, v\}$ of G is *maximal* if there is no other split pair $\{u', v'\}$ in G such that a split component of $\{u', v'\}$ contains both u and v.

The *decomposition tree* T of G is a rooted ordered tree recursively defined in four cases as follows. The nodes of T are of four types: S, P, Q, and R. Each node μ of T has an associated planar st-graph $\text{ske}(\mu)$, called the *skeleton* of μ. Also, μ is

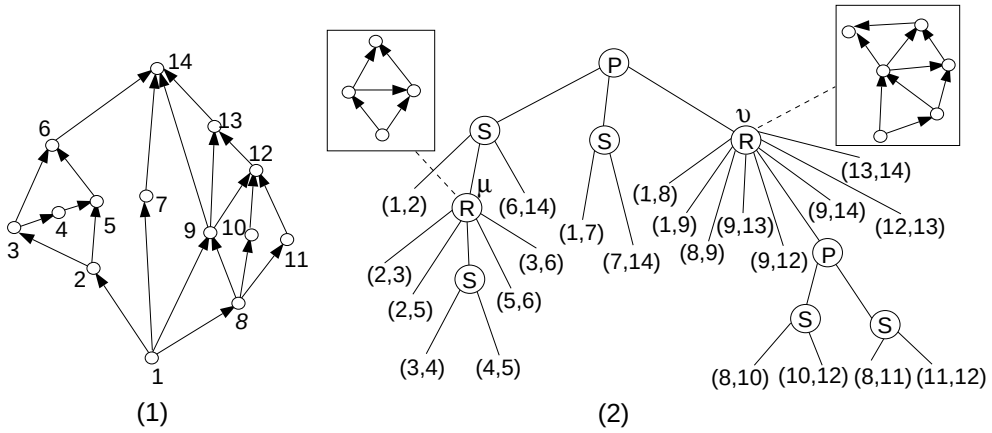


FIG. 6.1. *The tree in* (2) *is the decomposition tree of the graph in* (1).

associated with an edge in the skeleton of the parent ϕ of μ, called the *virtual edge* of μ in ske(ϕ).

Case Q. G is a single edge from s to t. Then T is a Q-node whose skeleton is G.

Case S. G is not biconnected. Let $c_1, \ldots, c_{k-1}$ with $k \geq 2$ be the cut vertices of G. Since G is a planar st-graph, each c_i is in exactly two blocks G_i and G_{i+1} with $s \in G_1$ and $t \in G_k$. Then T's root is an S-node μ, and ske(μ) consists of the chain $e_1, \ldots, e_k$, where the edge e_i goes from c_{i-1} to c_i, $c_0 = s$, and $c_k = t$.

Case P. $\{s, t\}$ is a split pair of G with k split components, where $k \geq 2$. Then T's root is a P-node μ, and ske(μ) consists of k parallel edges $e_1, \ldots, e_k$ from s to t.

Case R. Otherwise. Let $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ with $k \geq 1$ be the maximal split pairs of G. Let G_i be the union of the split components of $\{s_i, t_i\}$. Then T's root is an R-node μ, and ske(μ) is the simple graph obtained from G by replacing each G_i with an edge e_i from s_i to t_i. Note that adding the edge (s, t) to ske(μ) yields a simple triconnected graph.

Figure 6.1 illustrates the decomposition tree of G as well as the skeletons of μ and ν. In the last three cases, μ has children $\chi_1, \ldots, \chi_k$, in this order, such that each χ_i is the root of the decomposition tree of G_i. The virtual edge of χ_i is the edge e_i in ske(μ). G_i is called the *pertinent graph* pert(χ_i) of χ_i as well as the *expansion graph* of e_i. Note that G is the pertinent graph of T's root. Also, no child of an S-node is an S-node, and no child of a P-node is a P-node.

The *allocation nodes* of a vertex v of G are the nodes of T whose skeleton contains v; note that v has at least one allocation node.

LEMMA 6.2 (see [2]).

1. *T has $O(|G|)$ nodes and can be constructed in $O(|G|)$ time. The total number of edges of the skeletons stored at the nodes of T is $O(|G|)$.*
2. *The pertinent graphs of the children of μ can share only vertices of* ske(μ).
3. *If v is in* ske(μ), *then v is also in the pertinent graph of all ancestors of μ.*
4. *If v is a pole of* ske(μ), *then v is also in the skeleton of the parent of μ. If v is in* ske(μ) *but is not a pole of* ske(μ), *then v is not in the skeleton of any ancestor of μ.*
5. *The least common ancestor μ of the allocation nodes of v itself is an allocation node of v, called the* proper allocation node *of v. Also, if $v \notin \{s, t\}$, then μ is the only allocation node of v such that v is not a pole of* ske(μ).

6. *If $v \neq s, t$, then the proper allocation node of v is an R-node or S-node.*

For each non-S-node μ in T, $\text{pert}(\mu)$ is called a *block* of G [2], which differs from that in sections 4 and 5. For a block $B = \text{pert}(\mu)$, let $\text{node}(B) = \mu$. For an ancestor ϕ of $\text{node}(B)$, the *representative* of B in $\text{ske}(\phi)$ is the edge in $\text{ske}(\phi)$ whose expansion graph contains B.

Let μ be an R-node or P-node in T with children $\chi_1, \ldots, \chi_b$. For each $k \in \{1, \ldots, b\}$, let e_k be the virtual edge of χ_k in $\text{ske}(\mu)$. If χ_k is an S-node, $\text{pert}(\chi_k)$ is a chain consisting of two or more blocks. If χ_k is an R-node or P-node, $\text{pert}(\chi_k)$ is a single block. For each $k \in \{1, \ldots, b\}$, we say that the blocks in $\text{pert}(\chi_k)$ are *on edge* e_k. The *minor blocks* of $\text{pert}(\mu)$ are the blocks on e_1,... , the blocks on e_b.

6.2. Basic ideas. An *st-orientation* of a planar graph is an orientation of its edges together with an embedding such that the resulting digraph is a planar st-graph.

LEMMA 6.3 (see [1, 2]). *If an n-vertex simple planar graph has an st-orientation, then every embedding, where s and t are on the exterior face, of this graph can be obtained from this orientation through a sequence of $O(n)$ following operations:*

1. *Flip an R-node's skeleton around its poles.*
2. *Permute a P-node's children (and consequently their skeletons with respect to their common poles).*

Let $\{s, t\}$ be an edge of $\mathcal{G}$. Since $\mathcal{G}$ is a simple biconnected graph, we convert $\mathcal{G}$ to a planar st-graph in $O(n)$ time [12] for technical convenience. For the remainder of section 6, let T be the decomposition tree of $\mathcal{G}$.

The CFE algorithm processes the nodes of T in a bottom-up manner. It first processes the leaf nodes of T. When processing a node μ, for each $\mathcal{C}_i$ such that $\text{pert}(\mu)$ is the smallest block that intersects every set in $\mathcal{C}_i$, the algorithm looks for an embedding of $\text{pert}(\mu)$ that satisfies $\mathcal{C}_i$. If this is impossible, the algorithm outputs "no" and stops; otherwise, it continues on to process the next node of T. We note, in passing, that Theorem 3.8(2) is used when processing R-nodes.

Let μ be a node of T. T_μ denotes the subtree of T rooted at μ, and $\text{dep}(\mu)$ denotes the distance from T's root to μ. We need the following definitions:

1. $U_{i,j}$ is *contained* in $\text{pert}(\mu)$ if the vertices of $U_{i,j}$ are all in $\text{pert}(\mu)$; $U_{i,j}$ is *strictly contained* in $\text{pert}(\mu)$ if, in addition, no pole of $\text{pert}(\mu)$ is in $U_{i,j}$.
2. Let $\text{done}(U_{i,j})$ be the deepest node μ in T such that $U_{i,j}$ is strictly contained in $\text{pert}(\mu)$, if such a node exists. If no such μ exists, then $U_{i,j}$ contains a pole of $\mathcal{G}$, and let $\text{done}(U_{i,j})$ be T's root.
3. A family $\mathcal{C}_i$ *straddles* $\text{pert}(\mu)$ if at least one set in $\mathcal{C}_i$ is strictly contained in $\text{pert}(\mu)$ and at least one set in $\mathcal{C}_i$ has no vertex in $\text{pert}(\mu)$.
4. Let $\text{done}(\mathcal{C}_i)$ be the deepest node μ in T such that, for every $U_{i,j} \in \mathcal{C}_i$, at least one vertex of $U_{i,j}$ is in $\text{pert}(\mu)$.
5. Let $\text{sub}(\mu) = \{U_{i,j} \mid \text{done}(U_{i,j}) = \mu\}$ and $\text{fam}(\mu) = \{\mathcal{C}_i \mid \text{done}(\mathcal{C}_i) = \mu\}$.
6. If μ is a P-node or R-node, let $\text{xfam}(\mu) = \text{fam}(\mu) \cup (\cup_{\chi_k} \text{fam}(\chi_k))$ and $\text{xsub}(\mu) = \text{sub}(\mu) \cup (\cup_{\chi_k} \text{sub}(\chi_k))$, where χ_k ranges over all S-children of μ.

In a fixed embedding of a block B, the poles of B divide the boundary of its exterior face into two paths $\text{side}_1(B)$ and $\text{side}_2(B)$, called the two *sides* of B. $U_{i,j}$ is *two-sided* for B if both $\text{side}_1(B)$ and $\text{side}_2(B)$ intersect $U_{i,j}$. In particular, $U_{i,j}$ is two-sided for B if it contains a pole of B. $U_{i,j}$ is *side*-1 (respectively, *side*-2) for B if only $\text{side}_1(B)$ (respectively, $\text{side}_2(B)$) intersects $U_{i,j}$. Assume that B is a minor block of $\text{pert}(\mu)$ for some μ. Let e_k be the representative of B in $\text{ske}(\mu)$. In a fixed embedding of $\text{ske}(\mu)$, e_k separates two faces F and F'. When embedding $\text{pert}(\mu)$, we

can embed $\text{side}_1(B)$ toward either F or F', referred to as the two *orientations* of B in $\text{pert}(\mu)$.

A family $\mathcal{C}_i$ is *side*-0 (respectively, *side*-1 or *side*-2) *exterior-forcing* for B if $\text{done}(\mathcal{C}_i)$ is an ancestor of $\text{node}(B)$ in T and some $U_{i,j} \in \mathcal{C}_i$ strictly contained in B is two-sided (respectively, side-1 or side-2) for B. For $p = 0, 1, 2$, define the following.

1. $\text{ext}_p(B) = \min\{\text{dep}(\text{done}(\mathcal{C}_i)) \mid \mathcal{C}_i, 1 \le i \le q, \text{ is side-}p \text{ exterior-forcing for } B\}$ if at least one family in $\mathcal{M}$ is side-p exterior-forcing for B;
2. $\text{ext}_p(B) = \infty$ otherwise.

Assume $\text{ext}_p(B) \neq \infty$. Let $\mu = \text{node}(B), \phi_1, \phi_2, \ldots, \phi_h$ be the path in T from μ to ϕ_h, where $\text{dep}(\phi_h) = \text{ext}_p(B)$. For each $\ell \in \{1, \ldots, h-1\}$, the representative of B in $\text{ske}(\phi_\ell)$ must be an exterior edge in any satisfying embedding of $\text{ske}(\phi_\ell)$. In addition, if $p = 1$ or 2, $\text{side}_p(B)$ must be embedded toward the exterior face of the embedding of $\text{pert}(\phi_\ell)$.

Since (s, t) is an edge of $\mathcal{G}$, the root ρ of T is a P-node and has a child Q-node ϕ representing (s, t). A subtle difference between ρ and each nonroot node of T is that the two sides of $\mathcal{G} = \text{pert}(\rho)$ are actually on the same face. To eliminate this difference, we delete ϕ from T; afterward, if ρ has only one child, we further delete ρ from T. From here onward, T denotes this modified tree.

6.3. The CFE algorithm. The CFE algorithm processes T from the bottom up. A *ready* node μ of T is either (1) a leaf node or (2) a P-node or R-node such that the non-S-children of μ and the children of every S-child of μ all have been processed. The CFE algorithm processes the ready nodes of T in an arbitrary order. An S-node is processed when its parent is processed. We detail how to process μ as follows.

For the case where μ is a leaf node of T, note that $\text{pert}(\mu)$ is a single edge of $\mathcal{G}$. Since no $U_{i,j}$ is strictly contained in $\text{pert}(\mu)$, $\text{sub}(\mu) = \emptyset$. Also, each $\mathcal{C}_i \in \text{fam}(\mu)$ is satisfied by every embedding of $\mathcal{G}$. Therefore, we simply set $\text{ext}_p(\text{pert}(\mu)) = \infty$ for $p = 0, 1, 2$.

We next consider the case where μ is a non-leaf-ready node. Before μ is processed, an embedding of every minor block of $\text{pert}(\mu)$ is already fixed, except for a possible flip around its poles. Moreover, for each minor block B of $\text{pert}(\mu)$ and each $p \in \{0, 1, 2\}$, $\text{ext}_p(B)$ is known. When processing μ, the CFE algorithm checks whether some embedding Φ_μ of $\text{pert}(\mu)$ satisfies the following two conditions:

1. Φ_μ satisfies every $\mathcal{C}_i$ in $\text{xfam}(\mu)$.
2. For each $\mathcal{C}_i$ straddling $\text{pert}(\mu)$ and each $U_{i,j} \in \mathcal{C}_i$ strictly contained in $\text{pert}(\mu)$, at least one vertex of $U_{i,j}$ is embedded on the exterior face of Φ_μ. (*Remark.* This ensures the existence of an embedding of $\text{pert}(\text{done}(\mathcal{C}_i))$ satisfying $\mathcal{C}_i$ later.)

If no such Φ_μ exists, then $\mathcal{G}$ cannot satisfy $\mathcal{M}$, and the CFE algorithm outputs "no" and stops. Otherwise, it finds such a Φ_μ and fixes it except for a possible flip around its poles. It also computes $\text{ext}_p(\text{pert}(\mu))$ for $p = 0, 1, 2$.

To detail how to process μ, we classify the sets $U_{i,j}$ that intersect $\text{pert}(\mu)$ into four types and define a set $\text{img}(U_{i,j}, \mu)$ for each type as follows.

Type 1. $U_{i,j}$ contains at least one pole of $\text{ske}(\mu)$. Then $\text{done}(U_{i,j})$ is an ancestor of μ. Let $\text{img}(U_{i,j}, \mu) = \{v \in U_{i,j} \mid v \text{ is a vertex in } \text{ske}(\mu)\}$.

Type 2. $U_{i,j}$ contains at least one vertex but no pole of $\text{ske}(\mu)$. Then $\text{done}(U_{i,j}) = \mu$. Let $\text{img}(U_{i,j}, \mu)$ as in the case of type 1.

Type 3. $U_{i,j}$ is strictly contained in $\text{pert}(\chi)$ for some S-node child χ of μ, and $U_{i,j}$ contains at least one vertex in $\text{ske}(\chi)$. Then $\text{done}(U_{i,j}) = \chi$. Let $\text{img}(U_{i,j}, \mu)$ consist

of the virtual edge of χ in ske(μ).

Type 4. $U_{i,j}$ is strictly contained in a minor block B of pert(μ). Then done($U_{i,j}$) is node(B) or its descendent. Let img($U_{i,j}, \mu$) consist of the representative of B in ske(μ).

Each element of img($U_{i,j}, \mu$) is called an *image* of $U_{i,j}$ in ske(μ). The remainder of section 6.3 details how to process μ.

6.3.1. Processing an S-child of μ. When processing μ, for each S-child χ of μ, we need to find an embedding of pert(χ) satisfying certain conditions. We call this process the *S-procedure* and describe it below.

Let χ be an S-child of μ. Then ske(χ) is a path. Let $e_1, \ldots, e_b$ be the edges in ske(χ). For each $k \in \{1, \ldots, b\}$, let B_k be the expansion graph of e_k. Before the S-procedure is called on χ, the following requirements are met.

1. For each $k \in \{1, \ldots, b\}$, an embedding of B_k has been fixed, except for a possible flip around its poles.
2. For some integers $k \in \{1, \ldots, b\}$ and $p \in \{1, 2\}$, side$_p(B_k)$ is required to face either the left or the right side of ske(χ).

Our only choice for embedding pert(χ) is to flip $B_1, \ldots, B_b$ around their poles. We need to check whether, for some combination of flippings of $B_1, \ldots, B_b$, (1) the resulting embedding satisfies every $\mathcal{C}_i \in$ fam(χ) and (2) the second requirement above is met.

The S-procedure consists of the following five stages.

• Stage S1 constructs an auxiliary graph $D = (V_D, E_D)$ with $V_D = \{k_p \mid 1 \le k \le b,\ p = 1, 2\}$ as follows. For each $\mathcal{C}_i \in$ fam(χ), insert an arbitrary path P_i into D to connect all $k_p \in V_D$ such that, for some type-4 $U_{i,j} \in \mathcal{C}_i$, (a) img($U_{i,j}, \chi$) = $\{e_k\}$, and (b) $U_{i,j}$ is side-p for B_k. To avoid confusion, we call the elements of V_D *points* and the connected components of D *clusters*. Those points $k_p \in V_D$ such that side$_p(B_k)$ is required to face the left side of ske(χ) are called *L-points*. *R-points* are defined similarly. Note that, for each cluster C of D, all side$_p(B_k)$, where k_p ranges over all the points in C, must be embedded toward the same side of ske(χ). Also, each type-3 $U_{i,j}$ in $\mathcal{C}_i$ contains a vertex in ske(χ) which is on both sides of ske(χ). For this reason, such sets were not considered when constructing D.

• Stage S2 checks whether there is a cluster of D containing both an L-point and an R-point. If such a cluster exists, then S2 outputs "no" and stops. Suppose that no such cluster exists. If a cluster C contains an L-point (respectively, R-point), we call C an *L-cluster* (respectively, *R-cluster*).

• Stage S3 constructs another auxiliary graph $RD = (V_{RD}, E_{RD})$ from D as follows. The vertices of RD are the clusters of D. For each $k \in \{1, \ldots, b\}$, there is an edge $\{C_1, C_2\}$ in RD, where C_1 (respectively, C_2) is the cluster of D containing point k_1 (respectively, k_2). Note that RD may have self-loops.

• Stage S4 checks whether RD is bipartite. If it is not, then S4 outputs "no" and stops. Otherwise, for each connected component K of RD, the clusters in K can be uniquely partitioned into two independent subsets $V_{K,1}$ and $V_{K,2}$ of clusters. If $V_{K,1}$ or $V_{K,2}$ contains both an L-cluster and an R-cluster, S4 outputs "no" and stops. Otherwise, V_{RD} can be partitioned into two independent subsets V^L_{RD} and V^R_{RD} of clusters such that all L-clusters are in V^L_{RD} and all R-clusters are in V^R_{RD}. Let $V^L_D = \{k_p \mid k_p$ is in a cluster in $V^L_{RD}\}$ and $V^R_D = \{k_p \mid k_p$ is in a cluster in $V^R_{RD}\}$.

• Stage S5 embeds side$_p(B_k)$ toward the left side of ske(χ) for each $k_p \in V^L_D$.

Example 1. In Figure 6.2, pert(χ) has eight blocks $B_1, \ldots, B_8$. The left side of each B_k is side$_1(B_k)$. Also, fam(χ) = $\{\mathcal{C}_1, \ldots, \mathcal{C}_6\}$. An integer i in a small square on

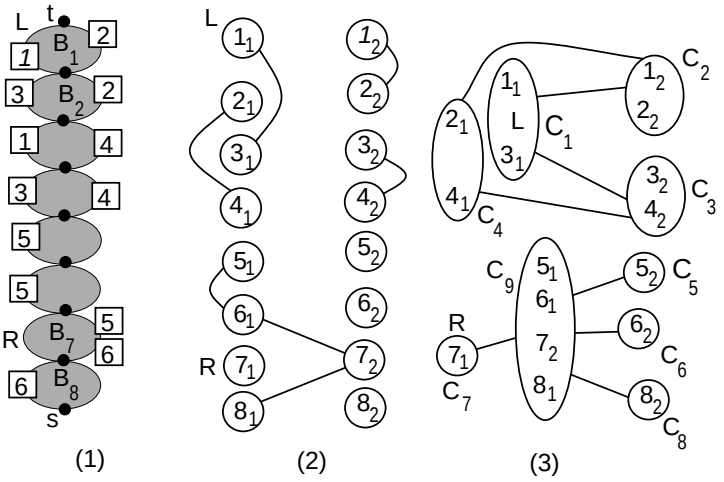


FIG. 6.2. *The graph in* (1) *is* pert(χ) *for an S-node* χ*, the graph in* (2) *is* D*, and that in* (3) *is* RD.

side$_p(B_k)$ for $p = 1$ or 2 indicates that k_p is on P_i. For example, the points on P_5 are 5_1, 6_1, and 7_2. The letter L is marked on side$_1(B_1)$, indicating that side$_1(B_1)$ must face left. The letter R is marked on side$_1(B_7)$, indicating that side$_1(B_7)$ must face right. D is shown in Figure 6.2(2). 1_1 is an L-point, while 7_1 is an R-point. RD is shown in Figure 6.2(3). C_1 is an L-cluster and C_7 is an R-cluster. RD is bipartite, and V_{RD} can be partitioned into $V^L_{RD} = \{C_1, C_4, C_9\}$ and $V^R_{RD} = \{C_2, C_3, C_5, C_6, C_7, C_8\}$. Thus $V^L_D = \{1_1, 2_1, 3_1, 4_1, 5_1, 6_1, 7_2, 8_1\}$ and $V^R_D = \{1_2, 2_2, 3_2, 4_2, 5_2, 6_2, 7_1, 8_2\}$. Flipping B_7 in Figure 6.2(1) gives a satisfying embedding of pert(χ). If 8_2 were also on P_5, there would be an edge $\{7_2, 8_2\}$ in D, which would cause C_9 and C_8 to be merged in RD with a self-loop attached to it. In that case, RD would not be bipartite, and the CFE algorithm would output "no."

6.3.2. μ is an R-node. In this case, adding the edge (s, t) to ske(μ) yields a simple triconnected graph. Thus the unique embedding of ske(μ) with both s and t on the exterior face is ske(μ) itself. Let $\chi_1, \ldots, \chi_b$ be the children of μ in T. For each $k \in \{1, \ldots, b\}$, let $B_{k,1}, \ldots, B_{k,s_k}$ be the minor blocks of pert(μ) in pert(χ_k). Note that $s_k = 1$ when χ_k is an R-node or P-node. To process μ, the CFE algorithm proceeds in five stages.

• Stage R1 first computes $\mathcal{C}'_i = \{\text{img}(U_{i,j}, \mu) \mid U_{i,j} \in \mathcal{C}_i\}$ for every $\mathcal{C}_i \in \text{fam}(\mu)$. Let $\mathcal{M}'(\mu)$ be the sequence of all $\mathcal{C}'_i$ with $\mathcal{C}_i \in \text{fam}(\mu)$. Then R1 calls Theorem 3.8(2) to solve the CFE problem on input ske(μ) and $\mathcal{M}'(\mu)$. If the output is "no," R1 outputs "no" and stops. Otherwise, for each $\mathcal{C}'_i$ in $\mathcal{M}'(\mu)$, there is a face F_i in ske(μ) whose boundary intersects each img$(U_{i,j}, \mu) \in \mathcal{C}'_i$. Note that F_i must be unique or else done($\mathcal{C}_i$) would be a descendent of μ, contradicting the fact that $\mathcal{C}_i \in \text{fam}(\mu)$.

• Stage R2 computes the minor block $B_{k,l}$ of pert(μ) strictly containing $U_{i,j}$ for each $\mathcal{C}_i \in \text{fam}(\mu)$ and each type-4 $U_{i,j} \in \mathcal{C}_i$. If $U_{i,j}$ is two-sided for $B_{k,l}$, either side of $B_{k,l}$ may be embedded toward the face F_i; otherwise, for some $p \in \{1, 2\}$, $U_{i,j}$ is side-p for $B_{k,l}$, and it requires that side$_p(B_{k,l})$ be embedded toward F_i.

• Stage R3 makes sure that, for every $\mathcal{C}_i$ straddling pert(μ) and for every $U_{i,j} \in \mathcal{C}_i$ strictly contained in pert(μ), a vertex in $U_{i,j}$ is embedded on the exterior face of pert(μ). This is done by checking whether the following statements are all false.

1. There are an exterior edge e_k of $\mathrm{ske}(\mu)$ and a minor block $B_{k,l}$ of $\mathrm{pert}(\mu)$ on e_k with $\max_{p\in\{1,2\}} \mathrm{ext}_p(B_{k,l}) < \mathrm{dep}(\mu)$; thus both $\mathrm{side}_1(B_{k,l})$ and $\mathrm{side}_2(B_{k,l})$ must be embedded toward the exterior face of $\mathrm{ske}(\mu)$.
2. There are an interior edge e_k of $\mathrm{ske}(\mu)$ and a minor block $B_{k,l}$ of $\mathrm{pert}(\mu)$ on e_k with $\min_{p\in\{0,1,2\}} \mathrm{ext}_p(B_{k,l}) < \mathrm{dep}(\mu)$; thus at least one of $\mathrm{side}_1(B_{k,l})$ and $\mathrm{side}_2(B_{k,l})$ must be embedded toward the exterior face of $\mathrm{ske}(\mu)$.
3. There is a $U_{i,j} \in \mathrm{sub}(\mu)$ with $\mathrm{dep}(\mathrm{done}(\mathcal{C}_i)) < \mathrm{dep}(\mu)$ (i.e., $\mathcal{C}_i$ straddles $\mathrm{pert}(\mu)$), and neither side of $\mathrm{ske}(\mu)$ contains an image in $\mathrm{img}(U_{i,j}, \mu)$.
4. There are an S-child χ_k of μ and a $U_{i,j} \in \mathrm{sub}(\chi_k)$ such that $\mathrm{dep}(\mathrm{done}(\mathcal{C}_i)) < \mathrm{dep}(\mu)$ and the virtual edge e_k of χ_k is an interior edge in $\mathrm{ske}(\mu)$.

If at least one statement above holds, R3 outputs "no" and stops. Otherwise, for each minor block $B_{k,l}$ of $\mathrm{pert}(\mu)$ such that $\mathrm{ext}_p(B_{k,l}) < \mathrm{dep}(\mu)$ for some $p \in \{1,2\}$, it requires that $\mathrm{side}_p(B_{k,l})$ be embedded toward the exterior face of $\mathrm{ske}(\mu)$. Note that, since statement 2 above is false, the representative e_k of $B_{k,l}$ in $\mathrm{ske}(\mu)$ must be an exterior edge of $\mathrm{ske}(\mu)$.

• Stage R4 first checks whether, for some minor block $B_{k,l}$ of $\mathrm{pert}(\mu)$, the orientation requirements imposed on $B_{k,l}$ in Stages R2 or R3 are in conflict. If they are, R4 outputs "no" and stops. Otherwise, for each R-child or P-child χ_k of μ, the minor block $\mathrm{pert}(\chi_k)$ can be oriented according to the requirements imposed on it or arbitrarily if no requirement was imposed on it. Afterward, for each S-child χ_k of μ, it calls the S-procedure on input χ_k together with the orientation requirements that were imposed on the minor blocks in $\mathrm{pert}(\chi_k)$ in Stages R2 or R3. If the S-procedure on a χ_k outputs "no," R4 outputs "no" and stops because $\mathrm{pert}(\chi_k)$ cannot be successfully embedded; otherwise, it has found a satisfying embedding of $\mathrm{pert}(\mu)$.

• Stage R5 computes $\mathrm{ext}_p(\mathrm{pert}(\mu))$ for $p = 0, 1, 2$ as follows. Let $\mathrm{xsub}'(\mu) = \{U_{i,j} \in \mathrm{xsub}(\mu) \mid \mathrm{dep}(\mathrm{done}(\mathcal{C}_i)) < \mathrm{dep}(\mu)\}$; i.e., $\mathrm{xsub}'(\mu)$ consists of all $U_{i,j} \in \mathrm{xsub}(\mu)$ such that $\mathcal{C}_i$ straddles $\mathrm{pert}(\mu)$. Partition $\mathrm{xsub}'(\mu)$ into A_0, A_1, A_2, where A_0 (respectively, A_1 or A_2) consists of all $U_{i,j} \in \mathrm{xsub}'(\mu)$ such that $U_{i,j}$ is two-sided (respectively, side-1 or side-2) for $\mathrm{pert}(\mu)$. For $i \in \{1,2\}$, let $\beta_i = \min_{p,B_{k,l}} \mathrm{ext}_p(B_{k,l})$, where p ranges over all integers in $\{0,1,2\}$ and $B_{k,l}$ ranges over all minor blocks on an edge of $\mathrm{side}_i(\mathrm{ske}(\mu))$. Then set

$$\mathrm{ext}_0(\mathrm{pert}(\mu)) = \min_{U_{i,j}\in A_0} \mathrm{dep}(\mathrm{done}(\mathcal{C}_i));$$

$$\mathrm{ext}_1(\mathrm{pert}(\mu)) = \min\left\{\beta_1, \min_{U_{i,j}\in A_1} \mathrm{dep}(\mathrm{done}(\mathcal{C}_i))\right\};$$

$$\mathrm{ext}_2(\mathrm{pert}(\mu)) = \min\left\{\beta_2, \min_{U_{i,j}\in A_2} \mathrm{dep}(\mathrm{done}(\mathcal{C}_i))\right\}.$$

This completes the processing of μ.

Example 2. In Figure 6.3, the circles denote the vertices in $\mathrm{ske}(\mu)$, where s and t are the poles of $\mathrm{pert}(\mu)$. An integer i in a small square at a side of a block $B_{k,l}$ indicates that a set in $\mathcal{C}_i$ has a vertex on that side of $B_{k,l}$. Also, $\mathrm{fam}(\mu) = \{\mathcal{C}_1, \mathcal{C}_2\}$. $\mathcal{C}_1 = \{U_{1,1}, \ldots, U_{1,4}\}$. $U_{1,1}$ is of type 3 and $\mathrm{img}(U_{1,1}, \mu) = \{e_3\}$. $U_{1,2}$ and $U_{1,3}$ are of type 4, $\mathrm{img}(U_{1,2}, \mu) = \{e_2\}$, and $\mathrm{img}(U_{1,3}, \mu) = \{e_4\}$. $U_{1,2}$ is two-sided for $B_{2,1}$. $U_{1,4}$ is of type 2 and $\mathrm{img}(U_{1,4}, \mu) = \{d\}$. $\mathcal{C}_2$ consists of $U_{2,1}$ and $U_{2,2}$, which are of type 4. $\mathrm{img}(U_{2,1}, \mu) = \{e_1\}$ and $\mathrm{img}(U_{2,2}, \mu) = \{e_2\}$. $\mathcal{C}_3$ is the only family straddling $\mathrm{pert}(\mu)$. $U_{3,1}$, $U_{3,2}$, and $U_{3,3}$ are the sets in $\mathcal{C}_3$ that intersect $\mathrm{pert}(\mu)$; the other sets in $\mathcal{C}_3$ are not shown in this figure. $U_{3,1}$ is of type 4 and $\mathrm{img}(U_{3,1}, \mu) = \{e_1\}$. $U_{3,2}$ is of type 2 and is two-sided for $\mathrm{pert}(\mu)$; $\mathrm{img}(U_{3,2}, \mu) = \{a, b, c\}$. Since $U_{3,3}$ is not strictly

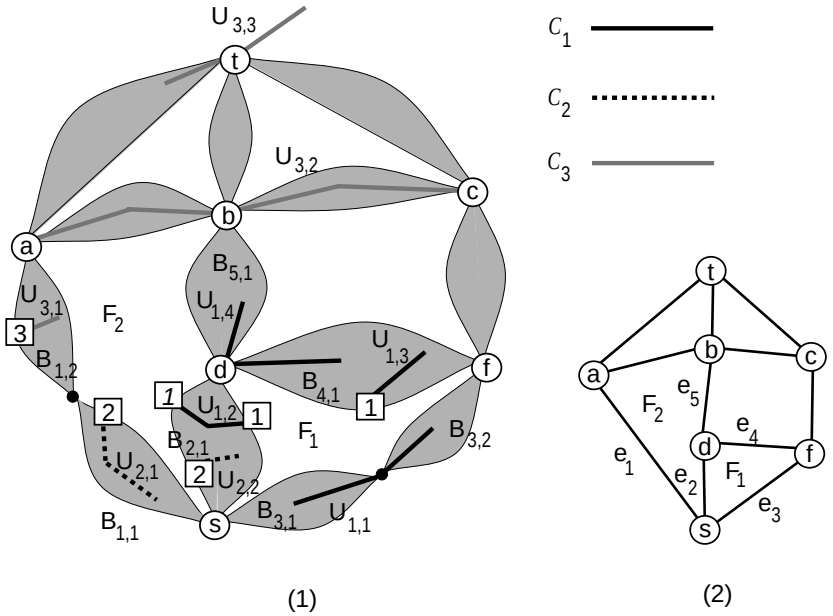


FIG. 6.3. *The graph in* (1) *is* pert(μ) *for an R-node* μ*, and the graph in* (2) *is* ske(μ).

contained in pert(μ), it is not tested during the processing of μ. Note that pert(μ) has a satisfying embedding as shown. For $i = 1, 2$, the boundary of F_i intersects each set in $\mathcal{C}_i$. The exterior face of pert(μ) contains an image of every set in $\mathcal{C}_3$ strictly contained in pert(μ). The side of $B_{4,1}$ on which 1 is marked must be embedded toward F_1. In contrast, whichever side of $B_{2,1}$ is embedded toward F_1, the boundary of F_1 intersects $U_{1,2}$. In the embedding of pert(μ), $\mathcal{C}_3$ is side-1 (respectively, side-0) exterior-forcing for pert(μ) because of $U_{3,1}$ (respectively, $U_{3,2}$).

6.3.3. μ is a P-node. In this case, ske(μ) consists of parallel edges $e_1, e_2, \ldots, e_b$ between its two poles with $b \geq 2$. Let $\chi_1, \ldots, \chi_b$ be the children of μ in T. For each $k \in \{1, \ldots, b\}$, let $B_{k,1}, \ldots, B_{k,s_k}$ be the minor blocks of pert(μ) in pert(χ_k). When embedding ske(μ), edges e_1 through e_b can be embedded in any order. The CFE algorithm first finds a proper embedding of ske(μ) in three stages.

• Stage P1 constructs an auxiliary graph $H = (V_H, E_H)$ with $V_H = \{e_1, \ldots, e_b\}$ by performing the following steps in turn for every $\mathcal{C}_i \in \text{fam}(\mu)$:

1. Compute $S_i = \cup_{U_{i,j}} \text{img}(U_{i,j}, \mu)$, where $U_{i,j}$ ranges over all type-3 or type-4 sets in $\mathcal{C}_i$. Let m_i be the number of edges in S_i. Then $m_i \geq 2$; otherwise, $\mathcal{C}_i$ would be in fam(χ_k) for some $k \in \{1, \ldots, b\}$.
2. If $m_i \geq 3$, then output "no" and stop since pert(μ) does not satisfy $\mathcal{C}_i$.
3. Insert edge $\{e_k, e_{k'}\}$ to H, where e_k and $e_{k'}$ are the two edges in S_i.

Note that, for each $\mathcal{C}_i \in \text{fam}(\mu)$, no set in $\mathcal{C}_i$ is of type 2, and each type-1 set in $\mathcal{C}_i$ contains a pole of pert(μ), which is on every face of all embeddings of ske(μ). For this reason, neither type-1 nor type-2 set in $\mathcal{C}_i$ is considered in the construction of H.

• Stage P2 checks whether both statements below are false in order to ensure that, for every $\mathcal{C}_i$ straddling pert(μ) and every $U_{i,j} \in \mathcal{C}_i$ strictly contained in pert(μ), a vertex in $U_{i,j}$ is embedded on the exterior face of pert(μ).

1. There is a minor block $B_{k,l}$ of pert(μ) with $\max_{p \in \{1,2\}} \text{ext}_p(B_{k,l}) < \text{dep}(\mu)$.

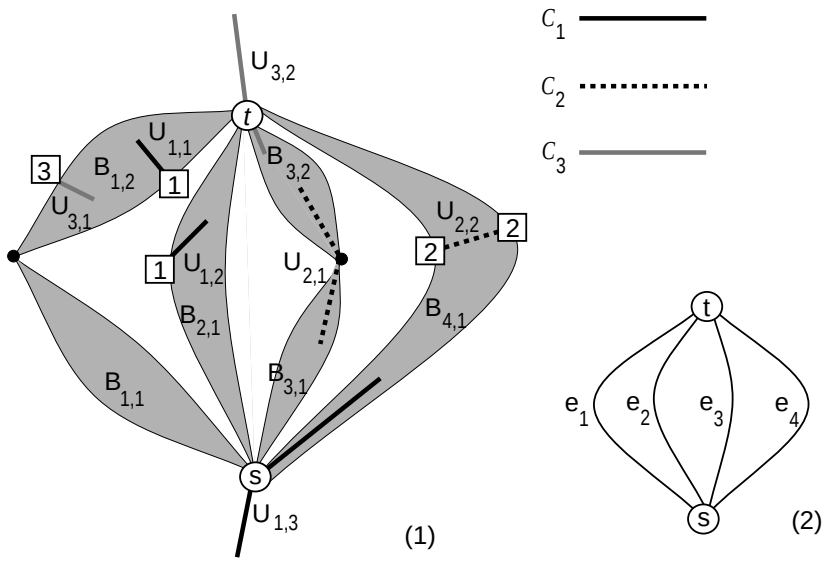


FIG. 6.4. *The graph in* (1) *is* $\mathrm{pert}(\mu)$ *for a P-node* μ*, and that in* (2) *is* $\mathrm{ske}(\mu)$.

2. There are at least three edges e_k in $\mathrm{ske}(\mu)$ such that (1) there is a minor block $B_{k,l}$ on e_k with $\min_{p\in\{0,1,2\}} \mathrm{ext}_p(B_{k,l}) < \mathrm{dep}(\mu)$ or (2) χ_k is an S-node and there exists $U_{i,j}$ in $\mathrm{sub}(\chi_k)$ with $\mathrm{dep}(\mathrm{done}(\mathcal{C}_i)) < \mathrm{dep}(\mu)$.

If statements 1 or 2 hold, P2 outputs "no" and stops. Otherwise, it marks each $e_k \in V_H$ for which statement 2(1) or 2(2) holds. Note that at most two $e_k \in V_H$ are marked, and each marked $e_k \in V_H$ must be an exterior edge in any satisfying embedding of $\mathrm{ske}(\mu)$.

• Stage P3 outputs "no" and stops if an $e_k \in V_H$ has degree at least 3 in H or a marked $e_k \in V_H$ has degree 2 in H. Otherwise, P3 finds and fixes an embedding of $\mathrm{ske}(\mu)$, where (1) each marked $e_k \in V_H$ is in the exterior face and (2) for every $\{e_k, e_{k'}\} \in E_H$, e_k and $e_{k'}$ form the boundary of a face. For each $\mathcal{C}_i \in \mathrm{fam}(\mu)$, let F_i be the face in the fixed embedding of $\mathrm{ske}(\mu)$ whose boundary is formed by the two edges in S_i. Note that, for each $U_{i,j} \in \mathcal{C}_i$, the boundary of F_i intersects $\mathrm{img}(U_{i,j}, \mu)$.

Next, the CFE algorithm tries to embed $\mathrm{pert}(\mu)$ based on the embedding of $\mathrm{ske}(\mu)$ fixed in Stage P3 through the same stages as Stages R2 through R5 in section 6.3.2 except that, in the stage corresponding to R5, $A_0 = \emptyset$ and the algorithm sets $\mathrm{ext}_0(\mathrm{pert}(\mu)) = \infty$. This completes the processing of μ.

Example 3. In Figure 6.4, $\mathrm{fam}(\mu) = \{\mathcal{C}_1, \mathcal{C}_2\}$. $\mathcal{C}_1 = \{U_{1,1}, U_{1,2}, U_{1,3}\}$. Both $U_{1,1}$ and $U_{1,2}$ are of type 4; $\mathrm{img}(U_{1,1}, \mu) = \{e_1\}$ and $\mathrm{img}(U_{1,2}, \mu) = \{e_2\}$. $U_{1,3}$ is of type 1 and need not be tested during the processing of μ. $\mathcal{C}_2 = \{U_{2,1}, U_{2,2}\}$. $U_{2,1}$ is of type 3 and $\mathrm{img}(U_{2,1}, \mu) = \{e_3\}$. $U_{2,2}$ is of type 4 and $\mathrm{img}(U_{2,2}, \mu) = \{e_4\}$. $\mathcal{C}_3$ is the only family straddling $\mathrm{pert}(\mu)$. $\{U_{3,1}$ and $U_{3,2}\}$ are the sets in $\mathcal{C}_3$ that intersect $\mathrm{pert}(\mu)$; the other sets in $\mathcal{C}_3$ are not shown in this figure. Since $U_{3,2}$ contains the pole t of $\mathrm{pert}(\mu)$, it is not tested during the processing of μ. $U_{3,1}$ is of type 4 and $\mathrm{img}(U_{3,1}, \mu) = \{e_1\}$. $V_H = \{e_1, e_2, e_3, e_4\}$ and $E_H = \{\{e_1, e_2\}, \{e_3, e_4\}\}$. Only e_1 is marked in graph H. Figure 6.4(2) shows an embedding of $\mathrm{ske}(\mu)$ that might be found and fixed in Stage P3. This embedding of $\mathrm{ske}(\mu)$ results in a satisfying embedding of $\mathrm{pert}(\mu)$ as shown. If either $\mathcal{C}_1$ had another set strictly contained in block $B_{4,1}$ or $\mathcal{C}_3$ had another set

strictly contained in $B_{2,1}$, then pert(μ) has no satisfying embedding.

This completes the description of the CFE algorithm. Its correctness follows from the above discussion and Lemma 6.3.

6.4. Implementation and analysis. We implement the CFE algorithm as follows. The nodes of T are identified by their preorder numbers. At each node $\mu \in T$, we store dep(μ) and the preorder number of the largest node in T_μ. Let $\chi_1, \ldots, \chi_b$ be the children of μ. The nodes in $T_{\chi_1}, \ldots, T_{\chi_b}$ form an ordered partition of the nodes in $T_\mu - \{\mu\}$. For a node ν, we can check whether ν is in T_μ in $O(1)$ time. If $\nu \in T_\mu$, we can find the subtree T_{χ_k} containing ν in $O(\log |\mathcal{G}|)$ time by a binary search of the children of μ. We equip T with a data structure which can be constructed in linear time and which outputs a least common ancestor query in $O(1)$ time [16, 27].

We also store ske(μ) at μ. Each μ has a pointer to its virtual edge in its parent's skeleton. For each nonpole vertex of ske(μ), we mark μ as its proper allocation node. This takes $O(|\mathcal{G}|)$ total time by Lemma 6.2(1). Each edge e of $\mathcal{G}$ has a pointer to the leaf node in T that represents e.

LEMMA 6.4. *Given $\mathcal{G}$, $\mathcal{M}$, and T, we can compute* fam(μ), sub(μ), done($\mathcal{C}_i$), *and* done($U_{i,j}$) *for all nodes μ of T, all $\mathcal{C}_i$ in $\mathcal{M}$, and all $U_{i,j}$ in $\mathcal{C}_i$ in $O(I)$ total time.*

Proof. For each vertex v of $\mathcal{G}$, let low(v) be the deepest allocation node of v in T. In $O(|\mathcal{G}|)$ time, we can compute low(v) for all vertices v of $\mathcal{G}$. For a set $U_{i,j} \in \mathcal{C}_i$, if a pole of $\mathcal{G}$ is in $U_{i,j}$, then done($U_{i,j}$) is the root of T; otherwise, done($U_{i,j}$) is the least common ancestor of all low(v) with $v \in U_{i,j}$. So done($U_{i,j}$) can be computed in $O(|U_{i,j}|)$ time. Let low($U_{i,j}$) be the deepest one among all low(v) with $v \in U_{i,j}$. We can compute low($U_{i,j}$) in $O(|U_{i,j}|)$ time. Since done($\mathcal{C}_i$) is the least common ancestor of all low($U_{i,j}$) with $U_{i,j} \in \mathcal{C}_i$, it can be computed in $O(|\mathcal{C}_i|)$ time. Thus, in $O(I)$ total time, we can compute done($U_{i,j}$) and done($\mathcal{C}_i$) for all $\mathcal{C}_i$ in $\mathcal{M}$ and all $U_{i,j}$ in $\mathcal{C}_i$. Afterward, in $O(I)$ total time, we can compute fam(μ) and sub(μ) for all nodes μ of T. □

After processing μ, the CFE algorithm records the following information:

1. the embedding of ske(μ);
2. ext$_p$(pert(μ)) for $p = 0$, 1, and 2;
3. the edges and vertices on side$_1$(ske(μ)) and side$_2$(ske(μ)), respectively;
4. an integer $p = 0$, 1, or 2 for each $U_{i,j} \in$ xsub(μ), indicating whether $U_{i,j}$ is two-sided, side-1, or side-2 for pert(μ), respectively.

The CFE algorithm processes a P-node or R-node μ with the five operations below.

Operation 1 uses $O(|U_{i,j}| + \log |\mathcal{G}|)$ time to determine the type of a given $U_{i,j}$ in xfam(μ) and finds img($U_{i,j}, \mu$) as follows. Let $\nu =$ done($U_{i,j}$).

Case 1. dep(ν) $\leq$ dep(μ). Then $U_{i,j}$ is of type 1 or 2 for pert(μ). $U_{i,j}$ is of type 1 if and only if it contains a pole of pert(μ). Also, img($U_{i,j}, \mu$) consists of all $v \in U_{i,j}$ which are also in ske(μ). Note that $v \in$ ske(μ) if and only if μ is the proper allocation node of v or v is a pole of pert(μ).

Case 2. dep(ν) = dep(μ) + 1, and ν is an S-node. Then $U_{i,j}$ is of type 3 for pert(μ). Also, img($U_{i,j}, \mu$) consists of the virtual edge of ν in ske(μ).

Case 3. Otherwise. Then $U_{i,j}$ is of type 4 for pert(μ). Also, img($U_{i,j}, \mu$) is the virtual edge of χ_k in ske(μ), where χ_k is the child of μ such that ν is in the subtree T_{χ_k}.

Operation 2 checks in $O(|U_{i,j}|)$ time whether a given $U_{i,j} \in$ xsub(μ) has a vertex on either side of pert(μ) after an embedding of pert(μ) is fixed. If $U_{i,j} \in$ sub(μ), we check whether a vertex in img($U_{i,j}, \mu$) is on either side of ske(μ). If $U_{i,j} \in$ sub(χ_k)

for an S-child χ_k of μ, we check whether the virtual edge e_k of χ_k is on either side of $\text{ske}(\mu)$.

Operation 3 uses $O(1)$ time to check whether a given $U_{i,j} \in \text{xsub}(\mu)$ is in $\text{xsub}'(\mu)$ by checking whether $\text{dep}(\text{done}(\mathcal{C}_i)) < \text{dep}(\mu)$.

Operation 4 checks whether a given $U_{i,j}$ is strictly contained in $\text{pert}(\mu)$ and, if so, further computes the minor block B of $\text{pert}(\mu)$ strictly containing $U_{i,j}$ in $O(|U_{i,j}| + \log|\mathcal{G}|)$ total time. For the first task, we check whether (1) $\nu = \text{done}(U_{i,j})$ is a descendent of μ, or (2) $\nu = \mu$ and $U_{i,j}$ contains no pole of $\text{pert}(\mu)$. For the second task, we first find the child χ_k of μ such that T_{χ_k} contains ν. If χ_k is not an S-node, $\text{pert}(\chi_k)$ is B; otherwise, B is $\text{pert}(\eta)$, where η is the child of χ_k such that T_η contains ν.

Operation 5 checks in $O(\log|\mathcal{G}|)$ time whether a given type-4 $U_{i,j}$ for $\text{pert}(\mu)$ is side-1, side-2, or two-sided for the minor block $B_{k,l}$ in $\text{pert}(\mu)$ strictly containing $U_{i,j}$. Let $\eta = \text{node}(B_{k,l})$ and $\nu = \text{done}(U_{i,j})$. Note that η has been processed. If $\eta = \nu$, this operation takes $O(1)$ time using the information stored for η. If ν is a descendent of η, the representative e of ν in $\text{ske}(\eta)$ can be found in $O(\log|\mathcal{G}|)$ time. Then it takes $O(\log|\mathcal{G}|)$ time to check whether e is on $\text{side}_1(\text{ske}(\eta))$ or $\text{side}_2(\text{ske}(\eta))$ using the information stored for η.

LEMMA 6.5.

1. $\{\text{xfam}(\mu) \mid \mu$ *is a P-node or R-node*$\}$ *is a partition of* $\{\mathcal{C}_1, \ldots, \mathcal{C}_q\}$.
2. $\{\text{xsub}(\mu) \mid \mu$ *is a P-node or R-node*$\}$ *is a partition of* $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_q$.
3. *Each input family* $\mathcal{C}_i$ *is processed exactly once.*
4. *Each input* $U_{i,j}$ *is processed at most twice, and the total time spent on processing* $U_{i,j}$ *is* $O(|U_{i,j}| + \log|\mathcal{G}|)$.

Proof. Statements 1 and 2 are straightforward. Statement 3 holds since each $\mathcal{C}_i$ is processed only when the node μ with $\mathcal{C}_i \in \text{xfam}(\mu)$ is processed. Each $U_{i,j}$ is processed once when the node μ with $U_{i,j} \in \text{xsub}(\mu)$ is processed and once when the node ϕ with $\mathcal{C}_i \in \text{xfam}(\phi)$ is processed. When $U_{i,j}$ is processed, we perform some of operations 1 through 5 on it. Since an operation takes $O(|U_{i,j}| + \log|\mathcal{G}|)$ time, statement 4 holds. □

We now bound the time of processing an R-node or P-node μ. Let $\text{xske}(\mu)$ be obtained from $\text{ske}(\mu)$ by replacing the virtual edge of each S-child χ_k of μ with $\text{ske}(\chi_k)$. Let n_μ be the number of vertices in $\text{xske}(\mu)$. Let $N_\mu = \sum_{\mathcal{C}_i \in \text{xfam}(\mu)} |\mathcal{C}_i|$. Recall that μ is processed using some of the following operations:

1. Process the sets $U_{i,j}$ in the families $\mathcal{C}_i \in \text{xfam}(\mu)$.
2. Call Theorem 3.8(2) on input $\text{ske}(\mu)$ and $\mathcal{M}'(\mu)$.
3. Call the S-procedure on χ_k for the S-children χ_k of μ.
4. Compute $\text{ext}_p(\text{pert}(\mu))$ for $p = 0$, 1, and 2.
5. Construct auxiliary graphs D, RD, and H, and operate on them.

Note that each $K \in \{D, RD, H\}$ is constructed and operated on in $O(|K|)$ total time. Since $\sum_K |K| \leq n_\mu$, where K ranges over all auxiliary graphs constructed during the processing of μ, it takes $O(n_\mu)$ total time to process the auxiliary graphs for μ. Therefore, the above operations take $O((n_\mu + N_\mu)\log I)$ time in total. By summing over all P-nodes and R-nodes μ of T, and by Theorem 3.8, Lemma 6.2(1), and Lemma 6.5, the CFE algorithm runs in the desired total time, completing the proof of Theorem 6.1.

7. Directions for further research. We have proved that the CFE problem can be solved in $O(I \log I)$ time for the special case where, for each input family $\mathcal{C}_i$, each set in $\mathcal{C}_i$ induces a connected subgraph of the input graph $\mathcal{G}$. One direction

for further research would be to reduce the running time to linear. Such a result might lead to substantial simplification of the SPQR tree decomposition or an entirely different data structure. Another worthy direction would be to solve more general cases in similar time bounds. Beyond these technical open problems, it would be of significance to find further applications of the CFE problem in addition to VLSI layout and topological inference as well as to identify novel and fundamental constrained planar embeddings.

Acknowledgments. We wish to thank the anonymous referees for many helpful suggestions.

REFERENCES

[1] G. D. Battista and R. Tamassia, *On-line graph algorithms with SPQR-trees*, Algorithmica, 15 (1996), pp. 302–318.

[2] G. D. Battista and R. Tamassia, *On-line planarity testing*, SIAM J. Comput., 25 (1996), pp. 956–997.

[3] D. Bienstock and C. L. Monma, *On the complexity of covering vertices by faces in a planar graph*, SIAM J. Comput., 17 (1988), pp. 53–76.

[4] K. Booth and G. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. System Sci., 13 (1976), pp. 335–379.

[5] J. Boyer and W. Myrvold, *Stop minding your P's and Q's: A simplified $O(n)$ planar embedding algorithm*, in Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 140–146.

[6] Z.-Z. Chen, M. Grigni, and C. H. Papadimitriou, *Map graphs*, J. ACM, 49 (2002), pp. 127–138.

[7] Z.-Z. Chen, X. He, and M.-Y. Kao, *Nonplanar topological inference and political-map graphs*, in Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 195–204.

[8] M. J. Egenhofer, *Reasoning about binary topological relations*, in Proceedings of Advances in Spatial Databases (SSD'91), Zurich, Switzerland, O. Gunther and H. J. Schek, eds., 1991, pp. 143–160.

[9] M. J. Egenhofer and J. Sharma, *Assessing the consistency of complete and incomplete topological information*, Geographical Systems, 1 (1993), pp. 47–68.

[10] G. Ehrlich, S. Even, and R. E. Tarjan, *Intersection graphs of curves in the plane*, J. Combin. Theory Ser. B, 21 (1976), pp. 8–20.

[11] R. E. Erickson, C. L. Monma, and A. F. Veinott, Jr., *Send-and-split method for minimum-concave-cost network flows*, Math. Oper. Res., 12 (1987), pp. 634–664.

[12] S. Even and R. E. Tarjan, *Computing an st-numbering*, Theoret. Comput. Sci., 2 (1976), pp. 339–344.

[13] G. N. Frederickson, *Using cellular graph embeddings in solving all pairs shortest paths problems*, J. Algorithms, 19 (1995), pp. 45–85.

[14] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[15] M. Grigni, D. Papadias, and C. H. Papadimitriou, *Topological inference*, in Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI), Vol. 1, Montreal, Canada, Morgan-Kaufmann, San Francisco, 1995, pp. 901–906.

[16] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.

[17] G. Kant and X. He, *Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems*, Theoret. Comput. Sci., 172 (1997), pp. 175–193.

[18] M.-Y. Kao, M. Fürer, X. He, and B. Raghavachari, *Optimal parallel algorithms for straight-line grid embeddings of planar graphs*, SIAM J. Discrete Math., 7 (1994), pp. 632–646.

[19] B. Mohar, *Embedding graphs in an arbitrary surface in linear time*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 392–397.

[20] J. Nešetřil and M. Rosenfeld, *Embedding graphs in Euclidean spaces, an exploration guided by Paul Erdős*, Geombinatorics, 6 (1997), pp. 143–155.

[21] T. Nishizeki and N. Chiba, *Planar Graphs: Theory and Algorithms*, North–Holland, Amsterdam, 1988.

[22] O. Ore, *The Four-Color Problem*, Academic Press, New York, 1967.

[23] D. Papadias and T. Sellis, *The qualitative representation of spatial knowledge in two-dimensional space*, Very Large Data Bases J., 4 (1994), pp. 479–516.
[24] C. H. Papadimitriou, D. Suciu, and V. Vianu, *Topological queries in spatial databases*, in Proceedings of the 15th ACM Symposium on Principles of Database Systems, ACM, New York, 1996, pp. 81–92.
[25] J. S. Provan, *Convexity and the Steiner tree problem*, Networks, 18 (1988), pp. 55–72.
[26] V. Ramachandran and J. Reif, *Planarity testing in parallel*, J. Comput. System Sci., 49 (1994), pp. 517–561.
[27] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, SIAM J. Comput., 17 (1988), pp. 1253–1262.
[28] D. D. Sleator and R. E. Tarjan, *Self-adjusting binary search tree*, J. ACM, 32 (1985), pp. 652–686.
[29] I. G. Tollis, *Graph drawing and information visualization*, ACM Comput. Surveys, 28 (1996), p. 19.
[30] H. Whitney, 2-*isomorphic graphs*, Amer. J. Math., 55 (1933), pp. 245–254.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 435–454

ACCELERATED SOLUTION OF MULTIVARIATE POLYNOMIAL SYSTEMS OF EQUATIONS*

B. MOURRAIN†, V. Y. PAN‡, AND O. RUATTA†

Abstract. We propose new Las Vegas randomized algorithms for the solution of a square nondegenerate system of equations, with well-separated roots. The algorithms use $\mathcal{O}(\delta\, 3^n D^2 \log(D)\log(b))$ arithmetic operations (in addition to the operations required to compute the normal form of the boundary monomials modulo the ideal) to approximate all real roots of the system as well as all roots lying in a fixed n-dimensional box or disc. Here D is an upper bound on the number of all complex roots of the system (e.g., Bezout or Bernshtein bound), δ is the number of real roots or the roots lying in the box or disc, and $\epsilon = 2^{-b}$ is the required upper bound on the output errors. For computing the normal form modulo the ideal, the efficient practical algorithms of [B. Mourrain and P. Trébuchet, in *Proceedings of the International Symposium on Symbolic and Algebraic Computation,* ACM, New York, 2000, pp. 231–238] or [J. C. Faugère, *J. Pure Appl. Algebra,* 139 (1999), pp. 61–88] can be applied. We also yield the bound $\mathcal{O}(3^n D^2 \log(D))$ on the complexity of counting the numbers of all roots in a fixed box (disc) and all real roots. For a large class of inputs and typically in practical computations, the factor δ is much smaller than D, $\delta = o(D)$. This improves by the order of magnitude the known complexity estimates of the order of at least $3^n D^4 + D^3 \log(b)$ or D^4, which so far are the record estimates even for the approximation of a single root of a system and for each of the cited counting problems, respectively. Our progress relies on proposing several novel techniques. In particular, we exploit the structure of matrices associated to a given polynomial system and relate it to the associated linear operators, dual space of linear forms, and normal forms of polynomials in the quotient algebra; furthermore, our techniques support the new nontrivial extension of the matrix sign and quadratic inverse power iterations to the case of multivariate polynomial systems, where we emulate the recursive splitting of a univariate polynomial into factors of smaller degree.

Key words. multivariate polynomials, systems of equations, quotient algebras, dual spaces, quasi-Toeplitz matrices, quasi-Hankel matrices, matrix sign iteration, quadratic power iteration

AMS subject classifications. 65H10, 68W30, 68W25, 68W20

PII. S0097539701385168

1. Introduction.

The classical problem of solving a multivariate polynomial system of equations is presently the subject of intensive research and one of the central practical and theoretical problems in the area of algebraic computation (see [21], [5], [32], [15].) It has major applications, for instance, to robotics, computer modelling and graphics, molecular biology, and computational algebraic geometry.

The oldest approach to the solution is the elimination method, reducing the problem to the computation of the associated resultant or its multiples. This classical method evolved in the old works by Bezout, Dixon, and Macaulay (see, e.g., [21], [45]) but later remained largely ignored by the researchers and algorithm designers until it was resurrected first by Chistov and Grigoriev [8], who designed a deterministic solution algorithm, then in a randomized approach by Canny [4], and later by

*Received by the editors February 20, 2001; accepted for publication (in revised form) September 19, 2002; published electronically February 4, 2003. Some results of this paper were presented at the 30th Annual ACM Symposium on Theory of Computing in 1998 and at the Smalefest (Hong Kong, 2000).

http://www.siam.org/journals/sicomp/32-2/38516.html

†Inria, Galaad, BP 93, 06902 Sophia-Antipolis, France (mourrain@sophia.inria.fr, oruatta@sophia.inria.fr).

‡Department of Mathematics and Computer Science, Lehman College, City University of New York, Bronx, NY 10468 (vpan@lehman.cuny.edu). The research of this author was supported by NSF grant CCR 9732206 and PSC CUNY award 62435-0031.

Giusti and Heintz [18] and has since become a very popular approach. One of the major further steps was the reduction of the solution of a multivariate polynomial system to matrix operations, in particular, by rational transformation of the original problem into a matrix eigenproblem (cf. [1], [16], [15], [27], [25], [10]).

The approach has been explored and extended by many researchers, has been exploited in the practice of algebraic computing, and has also supported the record asymptotic upper bound $\mathcal{O}^*(D^4)$ on the arithmetic computational complexity of the solution of a nondegenerate polynomial system having a finite number of roots [40]. Here and hereafter, $\mathcal{O}^*(s)$ stands for $\mathcal{O}(s \log^c s)$, c denoting a constant independent of s, and D is an upper bound on the number of roots of the given polynomial system. (For D, one may choose either the Bezout bound, $\prod_i d_i$, d_i denoting the maximum degree in the ith variable in all monomials of the system, or the Bernshtein bound, which is much smaller for sparse systems and equals the mixed volume of the associated Newton polytope, defined by the exponents of the monomials.) Even for many subproblems and related problems, no known algorithms support any better bound than $\mathcal{O}(D^4)$. This includes approximation of all real roots of a polynomial system (which is highly important due to applications to robotic and computer graphics), all its roots lying in a fixed n-dimensional box or disc, counting all roots in such a box or disc or all real roots, and even approximation of a single root. Some progress was achieved in [30], where a single root was approximated in $\mathcal{O}^*(3^n D^2)$ time, but under a strong restriction on the input polynomials.

Against this background, our new algorithms support the computational cost estimate of $\mathcal{O}^*(3^n D^2)$ for all of the subproblems listed above, that is, for both of the counting problems, the computation of a single root, all real roots, and all roots in a fixed box or disc. More precisely, our bound is $\mathcal{O}^*(\delta\, 3^n D^2)$ in the latter two cases, where δ is the number of real roots or roots in the selected box or disc, respectively. In practical applications, such a number is typically much less than D. The number of real roots grows as $\sqrt{D}$ for a large class of input systems [41]. See also the sparse case [24]. Thus, for all listed problems, we improve the known complexity estimates by an order of magnitude.

We have a reservation from a theoretical point of view; that is, our main algorithm relies on the known effective algorithms for the computation of the normal form of monomials on the boundary of the monomial basis (see section 4). These algorithms exploit structured matrices and in practice appear to run faster than our subsequent computations (see [17], [33]), but their known theoretical cost bounds are greater than the order of $e^{3n} D^3$ (see [22]).

Our paper addresses the problem of the asymptotic acceleration of the resolution stage, where the structure of the quotient algebra $\mathcal{A}$ (associated with the polynomial system) is already described by using the minimal number of parameters, that is, via the normal form of the monomials on the boundary of the basis. From a purely theoretical point of view, we have an alternative approach that avoids the normal-form algorithms at the price of using the order of $\mathcal{O}(12^n\, D^2)$ additional arithmetic operations [31]. This should be technically interesting because no other known approach yields this bound, but in this paper, we prefer to stay with our present, practically promising version, referring the reader to [31] on the cited theoretical approach. Our practically promising solution relies on fast computation of normal forms of polynomials modulo the ideal, based on the algorithm of [33]. Some limited amount of experimental evidence to the efficiency of this algorithm has been reported in [33], and further experimentation is ongoing.

Our algorithms approximate the roots numerically, and in terms of the required upper bound 2^{-b} (b is the bit precision) on the output errors of the computed solution, we obtain the running time estimate $\mathcal{O}(\log b)$ due to quadratic convergence of our algorithms. Within a constant factor, such an estimate matches the lower bound of [39] and enables us to yield a high output precision at relatively low cost; this gives us a substantial practical advantage versus the algorithms that reach only $\mathcal{O}(b)$ because the solution of a polynomial system is usually needed with a high precision. We achieve this by using the matrix sign iteration and the inverse quadratic iteration, both of which converge at a quadratic rate right from the start. All techniques and results can be extended to the case of sparse input polynomials (see Remark 3.16). In this case, the computation cost bounds become $\mathcal{O}(D\,C_{PolMult})$, where $C_{PolMult}$ is the cost of polynomial multiplication, which is small when the polynomials are sparse. (This cost depends on the degree of the polynomials and not only on an upper bound D on the number of roots.)

The factor 3^n is a substantial deficiency, of course, but it is still much less than D for the large and important class of input polynomials of degree higher than 3.

Our results require some other restrictions. First, we consider systems with simple roots or well-separated roots. In the presence of a cluster, a specific analysis is needed [43] and deserves additional work, which is not in the scope of this paper. Second, we need the existence of a nondegenerate linear form, which implies that the quotient algebra $\mathcal{A}$ is a Gorenstein algebra [12], [14]. This is the case in which the solution set is 0-dimensional and is defined by n equations. If we have more than n equations defining a 0-dimensional variety, we may take their n-random linear combination (see, e.g., [13]), which yields the required Gorenstein property, but this may introduce extra solutions that we will have to remove at the end. Finally, for approximation, our algorithms converge quadratically (using $\mathcal{O}(\log(b))$ steps) but require certain nondegeneracy assumptions (such as uniqueness of the minimum of the value of $|h(\zeta)|$, where ζ is a root and $h(x)$ is a polynomial). The latter assumptions can be ensured with a high probability by a random linear transformation of the variables. Even if these assumptions are barely satisfied, the slowdown of the convergence is not dramatic because the convergence is quadratic right from the start.

Similarly, we apply randomization to regularize the computations at the counting stages and for the auxiliary computation of the nondegenerate linear form in the dual space $\widehat{\mathcal{A}}$. Then again, nondegeneracy is ensured probabilistically and is verified in the subsequent computation. (That is, we stay under the Las Vegas probabilistic model, where failure may occur, with a small probability, but otherwise the correctness of the output is ensured.)

Some of our techniques should be of independent interest. In particular, we extend the theory of structured matrices to the ones associated to multivariate polynomials and show correlation among computations with such matrices and dual spaces of linear forms. We show some new nontrivial applications of the normal forms of polynomials of the quotient algebra. Furthermore, we establish new reduction from multivariate polynomial computations to some fundamental operations of linear algebra (such as the matrix sign iteration, the quadratic inverse power iteration, and the computation of Schur's complements).

Our progress has some technical similarity to the acceleration of the solution of linear systems of equations via fast matrix multiplication (in particular, we also rely on faster multiplication in the quotient algebra defined by the input polynomials) but even more so to the recent progress in the univariate polynomial rootfinding

via recursive splitting of the input polynomial into factors (cf. [6], [34], [36], [37]). Although recursive splitting into factors may be hard to even comprehend in the case of multivariate polynomial systems, this is exactly the basic step of our novel recursive process, which finally reduces our original problem to ones of small sizes. Of course, we could not achieve splitting in the original space of the variables, but we yield it in terms of idempotent elements of the associated quotient algebra (such elements represent the roots), and for this purpose we have to apply all of our advanced techniques. This approach generalizes the methods of [6] and [36] to the multivariate case. The only missing technical point of our extension of the univariate splitting construction of [36] is the balancing of the splitting, which was the most recent and elusive step in the univariate case (cf. [36], [37]). It is a major challenge to advance our approach to achieve balancing in our recursive splitting process even in the worst case (possibly by using the geometry of discriminant varieties) and, consequently, to approximate all of the roots of any specific polynomial system in $\mathcal{O}^*(3^n D^2 \log b)$ arithmetic time. Another goal is the computations in the dual space, as well as with structured matrices. The latter subject is of independent interest as well [44], [32].

Let us conclude this section with a high-level description of our approach. Our solution of polynomial systems consists of the following stages:

1. Compute a basic nondegenerate linear form on the quotient algebra $\mathcal{A}$ associated to a given system of polynomial equations.

2. Compute nontrivial idempotent elements of $\mathcal{A}$.

3. Recover the roots of the given polynomial system from the associated idempotents.

The quotient algebra $\mathcal{A}$ and the dual space of linear forms on it are defined and initially studied in section 2. Stage 1 is elaborated in section 4. Idempotents are computed by iterative algorithms of section 6. Section 7 shows how to recover or to count the roots efficiently when the idempotents are available. The computations are performed in the quotient algebra, and they are reduced to operations in the dual space by using the associated structured (quasi-Toeplitz and quasi-Hankel) matrices. In section 3, we define the classes of such matrices, show their correlation to polynomial computations, and exploit it to operate with such matrices faster. In section 5, we show how the combined power of the latter techniques and the ones developed for working in the dual space enable us to rapidly perform the basic operations in the quotient algebra and, consequently, the computations of sections 6 and 7.

Stage 1 contributes $O(3^n D^2 \log D)$ ops to the overall complexity bound, assuming that the normal form of the monomials on the boundary of a basis is known. The computation of a nontrivial idempotent at stage 2 has cost $O(3^n D^2 \log D \log b)$, which dominates the cost of the subsequent root counting or their recovery from the idempotents. The overall complexity depends on the number of idempotents that one has to compute, which in turn depends on the number δ of roots of interest. So far, we cannot utilize here the effective tools of balanced splitting, available in the similar situation for the univariate polynomial rootfinding. Thus, in the worst case, in each step we split out only a single root from the set of all roots, and then we need δ idempotents.

2. Definitions and preliminaries. Hereafter, $R = \mathbb{C}[x_1, \dots, x_n]$ is the ring of multivariate polynomials in the variables $x_1, \dots, x_n$, with coefficients in the complex field $\mathbb{C}$. $\mathbb{Z}$ is the set of integers, $\mathbb{N}$ is its subset of nonnegative integers, and $L = \mathbb{C}[x_1^{\pm}, \dots, x_n^{\pm}]$ is the set of Laurent polynomials with monomial exponents in $\mathbb{Z}^n$. For any $\mathbf{a} = (a_1, \dots, a_n) \in \mathbb{Z}^n$, $\mathbf{x}^{\mathbf{a}}$ is the monomial $\mathbf{x}^{\mathbf{a}} = x_1^{a_1} \cdots x_n^{a_n}$. $\lfloor E \rceil$ is the cardinality

(that is, the number of elements) of a finite subset E of $\mathbb{Z}^n$. "ops" will stand for "arithmetic operations" in the underlying coefficient ring or field.

2.1. Quotient algebra. To motivate and to demonstrate our study, we will next consider the univariate case, where we have a fixed polynomial $f \in \mathbb{C}[x]$ of degree d with d simple roots: $f(x) = f_d \prod_{i=1}^{d}(x-\zeta_i)$. The quotient algebra of residue polynomials modulo f, denoted by $\mathcal{A} = \mathbb{C}[x]/(f)$, is a vector space of dimension d. Its basis is $(1, x, \ldots, x^{d-1})$. Consider the Lagrange polynomials

$$\mathbf{e}_i = \prod_{j \neq i} \frac{x-\zeta_j}{\zeta_i - \zeta_j}.$$

One immediately sees that $\sum_i \mathbf{e}_i = 1$ and $\mathbf{e}_i\mathbf{e}_j \equiv \mathbf{e}_i(\mathbf{e}_i - 1) \equiv 0$ (for these two polynomials vanish at the roots of f). In other words, the Lagrange polynomials $\mathbf{e}_i$ are orthogonal idempotents in $\mathcal{A}$, and we have $\mathcal{A} = \sum_i \mathbb{C}\,\mathbf{e}_i$. Moreover, for any polynomial $a \in \mathcal{A}$, we also have $(a - a(\zeta_i))\mathbf{e}_i \equiv 0$, so that $\mathbf{e}_i$ is an eigenvector for the operator of multiplication by a in $\mathcal{A}$, for the eigenvalue $a(\zeta_i)$. These multiplication operators have a diagonal form in the basis $(\mathbf{e}_i)$ of $\mathcal{A}$. According to a basic property of Lagrange polynomials, we have $a \equiv \sum_i a(\zeta_i)\,\mathbf{e}_i(x)$ for any $a \in \mathcal{A}$. Therefore, the dual basis of $(\mathbf{e}_i)$ (formed by the coefficients of the $\mathbf{e}_i$ in this decomposition) consists of the linear forms associating to a its values at the points ζ_i. We will extend this approach to the case of multivariate polynomial systems, which, of course, will require substantial further elaboration and algebraic formalism. We refer the reader to [26], [27], [32], [42] for further details.

Let $f_1, \ldots, f_m$ be m polynomials of R, defining the polynomial system $f_1(x) = 0, \ldots, f_m(x) = 0$. Let I be the ideal generated by these polynomials, that is, the set of polynomial combinations $\sum_i f_i q_i$ of these elements. $\mathcal{A} = R/I$ denotes the quotient ring (algebra) defined in R by I, and $\equiv$ denotes the equality in $\mathcal{A}$. We consider the case in which the quotient algebra $\mathcal{A} = R/I$ *is of finite dimension* D *over* $\mathbb{C}$. This implies that the set of roots or solutions $\mathcal{Z}(I) = \{\zeta \in \mathbb{C}^n; f_1(\zeta) = \cdots = f_m(\zeta) = 0\}$ is finite: $\mathcal{Z}(I) = \{\zeta_1, \ldots, \zeta_d\}$ with $d \leq D$. Then we have a decomposition of the form

$$\mathcal{A} = \mathcal{A}_1 \oplus \cdots \oplus \mathcal{A}_d, \tag{1}$$

where $\mathcal{A}_i$ is a local algebra, for the maximal ideal $\mathbf{m}_{\zeta_i}$ defining the root ζ_i. From decomposition (1), we deduce that there exist orthogonal idempotents $\mathbf{e}_1, \ldots, \mathbf{e}_d$ satisfying

$$\mathbf{e}_1 + \cdots + \mathbf{e}_d \equiv 1 \text{ and } \mathbf{e}_i\,\mathbf{e}_j \equiv \begin{cases} 0 & \text{if } i \neq j, \\ \mathbf{e}_i & \text{if } i = j. \end{cases}$$

If $I = Q_1 \cap \cdots \cap Q_d$ is the minimal primary decomposition of I, we have $\mathbf{e}_i\mathcal{A} \sim R/Q_i$, where $\mathcal{A}_i = \mathbf{e}_i\,\mathcal{A}$ is a local algebra, for the maximal ideal $\mathbf{m}_{\zeta_i}$ defining the root ζ_i. Thus, to any root $\zeta \in \mathcal{Z}$, we associate an idempotent $\mathbf{e}_\zeta$.

2.2. Dual space. Let $\widehat{R}$ denote the dual of the $\mathbb{C}$-vector space R, that is, the space of linear forms

$$\begin{aligned} \Lambda : R &\to \mathbb{C}, \\ p &\mapsto \Lambda(p). \end{aligned}$$

(R will be the primal space for $\widehat{R}$.) Let us recall two celebrated examples, that is, the *evaluation at a fixed point* ζ,

$$\begin{aligned} \mathbf{1}_\zeta \,:\, & R \to \mathbb{C}, \\ & p \mapsto p(\zeta), \end{aligned}$$

and the map

$$\begin{aligned} (\mathbf{d}^{\mathbf{a}} = (\mathbf{d}_1)^{a_1} \cdots (\mathbf{d}_n)^{a_n}) : R &\to \mathbb{C} \\ p &\mapsto \frac{1}{\prod_{i=1}^n a_i!}\, (d_{x_1})^{a_1} \cdots (d_{x_n})^{a_n}\, (p)(0), \end{aligned} \tag{2}$$

where $\mathbf{a} = (a_1, \ldots, a_n)$ is any vector from $\mathbb{N}^n$ and d_{x_i} is the partial derivative with respect to the variable x_i. For any $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{N}^n$, we have

$$\mathbf{d}^{\mathbf{a}}(\mathbf{x}^{\mathbf{b}}) = \begin{cases} 1 & \text{if } \forall i, a_i = b_i, \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, $(\mathbf{d}^{\mathbf{a}})_{\mathbf{a}\in\mathbb{N}^n}$ is the dual basis of the primal monomial basis. Thus we decompose any linear form $\Lambda \in \widehat{R}$ as

$$\Lambda = \sum_{\mathbf{a}\in\mathbb{N}^n} \Lambda(\mathbf{x}^{\mathbf{a}})\, \mathbf{d}^{\mathbf{a}}. \tag{3}$$

Hereafter, *we will identify* $\widehat{R}$ *with* $\mathbb{C}[[\mathbf{d}_1, \ldots, \mathbf{d}_n]]$. The map $\Lambda \to \sum_{\mathbf{a}\in\mathbb{N}^n} \Lambda(\mathbf{x}^{\mathbf{a}})\, \mathbf{d}^{\mathbf{a}}$ defines a one-to-one correspondence between the set of linear forms Λ and the set $\mathbb{C}[[\mathbf{d}_1, \ldots \mathbf{d}_n]] = \mathbb{C}[[\mathbf{d}]] = \{\sum_{\mathbf{a}\in\mathbb{N}^n} \lambda_{\mathbf{a}} \mathbf{d}_1^{a_1} \cdots \mathbf{d}_n^{a_n}\}$ of polynomials in the variables $\mathbf{d}_1, \ldots, \mathbf{d}_n$.

The evaluation at 0 corresponds to the constant 1 under this definition. It will also be denoted by $\delta_0 = \mathbf{d}^0$.

We will denote by $\widehat{\mathcal{A}}$ and also by $I^\perp$ the subspace of $\widehat{R}$ made of those linear forms that vanish on the ideal I.

We now define multiplication of a linear form by a polynomial ($\widehat{R}$ is an R-module) as follows. For any $p \in R$ and $\Lambda \in \widehat{R}$, we write

$$\begin{aligned} p \star \Lambda : R &\to \mathbb{C}, \\ q &\mapsto \Lambda(p\, q). \end{aligned}$$

For any pair of elements $p \in R$ and $a \in \mathbb{N},\ a > 1$, we have

$$(d_{x_i})^a\, (x_i\, p)(0) = a\, (d_{x_i})^{a-1}\, p(0).$$

Consequently, for any pair $(p, \mathbf{a})$, $p \in R$, $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{N}^n$ (where $a_i \neq 0$ for a fixed i), we obtain

$$\begin{aligned} x_i \star \mathbf{d}^{\mathbf{a}}(p) &= \mathbf{d}^{\mathbf{a}}(x_i\, p) \\ &= \mathbf{d}_1^{a_1} \cdots \mathbf{d}_{i-1}^{a_{i-1}} \mathbf{d}_i^{a_i - 1} \mathbf{d}_{i+1}^{a_{i+1}} \cdots \mathbf{d}_n^{a_n}(p); \end{aligned}$$

that is, x_i acts as the *inverse* of $\mathbf{d}_i$ in $\mathbb{C}[[\mathbf{d}]]$. For this reason, such a representation is referred to as the *inverse systems* (see, for instance, [23]). If $a_i = 0$, then $x_i \star \mathbf{d}^{\mathbf{a}}(p) = 0$, which allows us to redefine the product $p \star \Lambda$ as follows.

PROPOSITION 2.1. *For any pair $p, q \in R$ and any $\Lambda(\mathbf{d}) \in \mathbb{C}[[\mathbf{d}]]$, we have*

$$p \star \Lambda(q) = \Lambda(p\,q) = \pi_+(p(\mathbf{d}^{-1})\,\Lambda(\mathbf{d}))(q),$$

where π_+ is the projection mapping Laurent series onto the space generated by the monomials in $\mathbf{d}$ with positive exponents.

This yields the following algorithm.

ALGORITHM 2.2. *For any polynomial $p \in \langle \mathbf{x}^\alpha \rangle_{\alpha \in E}$ and a vector $[\Lambda(\mathbf{x}^\beta)]_{\beta \in E+F}$, compute the vector $[p \star \Lambda(\mathbf{x}^\beta)]_{\beta \in F}$ as follows:*

- *Write $\tilde{\Lambda}(\mathbf{d}) = \sum_{\beta \in E+F} \Lambda(\mathbf{x}^\beta)\,\mathbf{d}^\beta$.*
- *Compute the product $\rho(\mathbf{d}) = p(\mathbf{d}^{-1})\tilde{\Lambda}(\mathbf{d})$ in $\mathbb{C}[\mathbf{d}, \mathbf{d}^{-1}]$.*
- *Keep the coefficients ρ_α of $\mathbf{d}^\alpha$ for $\alpha \in F$.*

3. Quasi-Toeplitz and quasi-Hankel matrices. In this section, we describe the structure of the matrices and some tools that we will use for our algorithm design.

Let us recall first the known arithmetic complexity bounds for polynomial multiplication (see [2, pp. 56–64]), which is the basic step of our subsequent algorithms. Let $C_{PolMult}(E, F)$ denote the number of ops (that is, of arithmetic operations) required for the multiplication of a polynomial with support in E by a polynomial with support in F.

THEOREM 3.1. *Let $E + F = \{\alpha^i = (\alpha_1^{(i)}, \ldots, \alpha_n^{(i)}),\ i = 1, \ldots, N\}$ with $|\alpha^{(i)}| = \sum_j \alpha_j^{(i)} = d_i$ for $i = 1, \ldots, N$ and $d = \max_i(d_i)$. Let $C_{K;Eval}(G)$ ops suffice to evaluate a polynomial with a support G on a set of K points. Then we have*

$$C_{PolMut}(E, F) = \mathcal{O}\left(C_{N;Eval}(E) + C_{N;Eval}(F) + N\left(\log^2(N) + \log(d)\right)\right).$$

Proof. Apply the evaluation-interpolation techniques to multiply the two polynomials (cf. [2]). That is, first evaluate the input polynomials on a fixed set of N points, then multiply pairwise the computed values to obtain the values of the product on the same set, and finally interpolate from these values and compute the coefficients of the product by applying the (sparse) polynomial interpolation algorithm (cf. [2]). By summarizing the computational cost estimates, we obtain the theorem. □

For special sets E and F, we have better bounds.

THEOREM 3.2. *Let $E_d = [0, \ldots, d-1] \subset \mathbb{N}$. Then*

$$C_{PolMult}(E_d, E_d) = \mathcal{O}(d \log(d)).$$

THEOREM 3.3. *Let $E_{\mathbf{c}} = \{(\alpha_1, \ldots, \alpha_n)\ ;\ 0 \le \alpha_i \le c_i - 1\}$, $E_{\mathtt{d}} = \{(\beta_1, \ldots, \beta_n)\ ;\ 0 \le \beta_i \le d_i - 1\}$, $c = \max\{c_1, \ldots, c_n\}$, and $d = \max\{d_1, \ldots, d_n\}$. Then we have*

$$C_{PolMult}(E_{\mathbf{c}}, E_{\mathtt{d}}) = \mathcal{O}(M \log(M)),$$

where $M = f^n$ and $f = c + d + 1$.

THEOREM 3.4. *Let $E_{f,n}$ be the set of exponents having total degree at most f in n variables. Then*

$$C_{PolMult}(E_{c,n}, E_{d,n}) = \mathcal{O}(T \log^2(T)),$$

where $T = \binom{n+c+d}{n}$ is the number of monomials of degree at most $c+d$ in n variables.

REMARK 3.5. *Theorems 3.1 and 3.3 correspond, respectively, to lattice points in a product of intervals and in the scaled standard simplex and can be extended to the computations over any ring of constants (rather than over the complex field) at*

the expense of increasing their complexity bounds by at most the factors of $\log\log(N)$ *or* $\log\log(M)$*, respectively* [2]. *Theorem* 3.4 *can be extended similarly to any field of constants having characteristic* 0.

Next, by following [32], [31], we will extend the definitions of Toeplitz and Hankel matrices to the multivariate case. As we will see, these structures are omnipresent when we solve polynomial systems.

DEFINITION 3.6. *Let* E *and* F *be two finite subsets of* $\mathbb{N}^n$*, and let* $M = (m_{\alpha,\beta})_{\alpha\in E,\beta\in F}$ *be a matrix whose rows are indexed by the elements of* E *and columns by the elements of* F*. Let* $\mathtt{i}$ *denote the* i*th basis coordinate vector of* $\mathbb{N}^n$*.*

- $M = [m_{\alpha,\beta}]_{\alpha\in E,\beta\in F}$ *is an* (E,F) quasi-Toeplitz *matrix if and only if, for all* $\alpha \in E, \beta \in F$*, the entries* $m_{\alpha,\beta} = t_{\alpha-\beta}$ *depend only on* $\alpha - \beta$*, that is, if and only if, for* $i = 1,\dots,n$*, we have* $m_{\alpha+\mathtt{i},\beta+\mathtt{i}} = m_{\alpha,\beta}$*, provided that* $\alpha, \alpha+\mathtt{i} \in E; \beta, \beta+\mathtt{i} \in F$*; such a matrix* M *is associated with the polynomial* $T_M(\mathbf{x}) = \sum_{\mathbf{u}\in E+F} t_{\mathbf{u}}\,\mathbf{x}^{\mathbf{u}}$.
- M *is an* (E,F) quasi-Hankel *matrix if and only if, for all* $\alpha \in E, \beta \in F$*, the entries* $m_{\alpha,\beta} = h_{\alpha+\beta}$ *depend only on* $\alpha+\beta$*, that is, if and only if, for* $i = 1,\dots,n$*, we have* $m_{\alpha-\mathtt{i},\beta+\mathtt{i}} = m_{\alpha,\beta}$ *provided that* $\alpha, \alpha-i \in E; \beta, \beta+\mathtt{i} \in F$*; such a matrix* M *is associated with the Laurent polynomial* $H_M(\mathbf{d}) = \sum_{\mathbf{u}\in E-F} h_{\mathbf{u}}\mathbf{d}^{\mathbf{u}}$.

For $E = [0,\dots,m-1]$ and $F = [0,\dots,n-1]$ (resp., $F = [-n+1,\dots,0]$), Definition 3.6 turns into the usual definition of Toeplitz (resp., Hankel) matrices (see [2]). Quasi-Toeplitz matrices have also been studied under the name of multilevel Toeplitz matrices (see, e.g., [44]) in the restricted special case, where the sets E and F are rectangular (i.e., a product of intervals). For our study of polynomial systems of equations, using the latter restricted case is not sufficient, and our more general definitions are required.

The definitions can be extended immediately to all subsets E, F of $\mathbb{Z}^n$ if we work with the Laurent polynomials.

The classes of quasi-Toeplitz and quasi-Hankel matrices can be transformed into each other by means of multiplication by the reflection matrix, having ones on its antidiagonal and zeros elsewhere.

DEFINITION 3.7. *Let* $\pi_E : L \to L$ *be the projection map such that* $\pi_E(\mathbf{x}^\alpha) = \mathbf{x}^\alpha$ *if* $\alpha \in E$ *and* $\pi_E(\mathbf{x}^\alpha) = 0$ *otherwise. Also let* $\pi_E : \mathbb{C}[[\mathbf{d}]] \to \mathbb{C}[[\mathbf{d}]]$ *denote the projection map such that* $\pi_E(\mathbf{d}^\alpha) = \mathbf{d}^\alpha$ *if* $\alpha \in E$ *and* $\pi_E(\mathbf{d}^\alpha) = 0$ *otherwise.*

We can describe the quasi-Toeplitz and quasi-Hankel operators in terms of polynomial multiplication (see [30], [29]), and the next proposition reduces multiplication of an (E,F) quasi-Toeplitz (resp., quasi-Hankel) matrix by a vector $\mathbf{v} = [v_\beta] \in \mathbb{C}^F$ to (Laurent's) polynomial multiplication.

PROPOSITION 3.8. *The matrix* M *is an* (E,F) *quasi-Toeplitz (resp., an* (E,F) *quasi-Hankel) matrix if and only if it is the matrix of the operator* $\pi_E \circ \mu_{T_M} \circ \pi_F$ *(resp.,* $\pi_E \circ \mu_{H_M} \circ \pi_F$*), where, for any* $p \in L$*,* $\mu_p : q \mapsto p\,q$ *is the operator of multiplication by* p *in* L*.*

Proof (see [29]). We will give a proof only for an (E,F) quasi-Toeplitz matrix $M = (M_{\alpha,\beta})_{\alpha\in E,\beta\in F}$. (The proof is similar for a quasi-Hankel matrix.) The associated polynomial is $T_M(\mathbf{x}) = \sum_{\mathbf{u}\in E+F} t_{\mathbf{u}}\mathbf{x}^{\mathbf{u}}$. For any vector $\mathbf{v} = [v_\beta] \in \mathbb{C}^F$, let $v(\mathbf{x})$ denote

the polynomial $\sum_{\beta\in F} v_\beta \mathbf{x}^\beta$. Then

$$T_M(\mathbf{x})\, v(\mathbf{x}) = \sum_{\mathbf{u}\in E+F,\beta\in F} \mathbf{x}^{\mathbf{u}+\beta}\, t_{\mathbf{u}}\, v_\beta$$
$$= \sum_{\alpha=\mathbf{u}+\beta\in E+2\,F} \mathbf{x}^\alpha \left(\sum_{\beta\in F} t_{\alpha-\beta}\, v_\beta\right),$$

where we assume that $t_{\mathbf{u}} = 0$ if $\mathbf{u} \notin E + F$. Therefore, for $\alpha \in E$, the coefficient of $\mathbf{x}^\alpha$ equals

$$\sum_{\beta\in F} t_{\alpha-\beta}\, v_\beta = \sum_{\beta\in F} M_{\alpha,\beta}\, v_\beta,$$

which is precisely the coefficient α of $M\mathbf{v}$. □

ALGORITHM 3.9. *Multiplication of the (E, F) quasi-Toeplitz (resp., quasi-Hankel) matrix $M = (M_{\alpha,\beta})_{\alpha\in E,\beta\in F}$ by a vector $\mathbf{v} = [v_\beta] \in \mathbb{C}^F$:*

- *multiply the polynomials $T_M = \sum_{\mathbf{u}\in E+F} t_{\mathbf{u}}\, \mathbf{x}^{\mathbf{u}}$ (resp., $H_M(\mathbf{d}) = \sum_{\mathbf{u}\in E-F} h_{\mathbf{u}}\mathbf{d}^{\mathbf{u}}$) by $v(\mathbf{x}) = \sum_{\beta\in F} v_\beta \mathbf{x}^\beta$ (resp., $v(\mathbf{d}^{-1}) = \sum_{\beta\in F} v_\beta \mathbf{d}^{-\beta}$),*
- *and output the projection of the product on $\mathbf{x}^E$ (resp., $\mathbf{d}^E$).*

DEFINITION 3.10. *$C_{PolMult}(E, F)$ denotes the number of ops required to multiply a polynomial with a support in E by a polynomial with a support in F.*

Clearly, Algorithm 3.9 uses $C_{PolMult}(E+F, F)$ (resp., $C_{PolMult}(E-F, -F)$) ops.

PROPOSITION 3.11.

(a) *An (E, F) quasi-Hankel (resp., an (E, F) quasi-Toeplitz) matrix M can be multiplied by a vector by using $\mathcal{O}(N \log^2(N) + N \log(d) + C_{M,N})$ ops, where $d = \deg H_M$ (resp., $\deg T_M$) , $N = \lfloor E - 2\,F \rceil$ (resp., $\lfloor E + 2F \rceil$), and $C_{M,N}$ denotes the cost of the evaluation of all monomials of the polynomial H_M (resp., T_M) on a fixed set of N points.*

(b) *In particular, the ops bound becomes $\mathcal{O}(M \log(M))$, where $E + F = E_{\mathbf{c}}, F = E_{\mathbf{d}}$ and $E_{\mathbf{c}}, E_{\mathbf{d}}$ and $M = (c + d + 1)^n$ are defined as in Theorem 3.3, whereas*

(c) *the bound turns into $\mathcal{O}(T \log^2(T))$, where $E + F = E_{c,n}, F = E_{d,n}$ and $E_{c,n}, E_{d,n}$, and $T = \binom{n+c+d}{n}$ are defined as in Theorem 3.4.*

Proof. Reduce the problem to computing the product of the two polynomials $H_M(\mathbf{x})$ (resp., $T_M(\mathbf{x})$) and $V(\mathbf{x})$, and then apply Theorems 3.1–3.4. □

Applying these results, we can bound the number of ops in Algorithm 2.2 as follows.

PROPOSITION 3.12. *For any polynomial $p \in R$ with support in E and any vector $[\Lambda(\mathbf{x}^\alpha)]_{\alpha\in E+F}$ (with $\Lambda \in \widehat{R}$), the vector $[p \star \Lambda(\mathbf{x}^\beta)]_{\beta\in F}$ can be computed in $\mathcal{O}(\lfloor E + F \rceil \log^2(\lfloor E + F \rceil))$ ops.*

Once we have a fast matrix-by-vector multiplication, a nonsingular linear system of equations can also be solved quickly by means of the conjugate gradient algorithm, which is based on the following theorem [19, section 10.2].

THEOREM 3.13. *Let $W\,\mathbf{v} = \mathbf{w}$ be a nonsingular linear system of N equations. Then N multiplications of each of the matrices W and W^T by vectors and $\mathcal{O}(N^2)$ additional ops suffice to compute the solution $\mathbf{v}$ to this linear system.*

Note that W^T is a quasi-Toeplitz (resp., quasi-Hankel) matrix if W is, and then both matrices can be multiplied by a vector quickly (see Proposition 3.11). Therefore, in the cases of quasi-Toeplitz and quasi-Hankel matrices W, Theorem 3.13 yields a

fast algorithm for solving the linear system $W\,\mathbf{v} = \mathbf{w}$. We will also need the following related result.

THEOREM 3.14 (see [32]). *Let W be an $N \times N$ real symmetric or Hermitian matrix. Let S be a fixed finite set of complex numbers. Then there is a randomized algorithm that selects N random parameters from the set S independently of each other (under uniform probability distribution on S) and either fails with a probability of at most $\frac{(N+1)N}{2\lfloor S\rceil}$ or performs $\mathcal{O}(N)$ multiplications of the matrix W by vectors and $\mathcal{O}(N^2\log(N))$ other ops to compute the rank and the signature of W.*

Hereafter, random selection of elements of a set S as in Theorem 3.14 will be called *sampling*.

Proof. To support the claimed estimate, we first tridiagonalize the matrix W by the Lanczos randomized algorithm [2, pp. 118–119], which involves an initial vector of dimension N and fails with a probability of $\frac{(N+1)N}{2\lfloor S\rceil}$ if the N coordinates of the vector have been sampled at random from the set S. The above bound on the failure probability and the cost bound of $\mathcal{O}(N)$ multiplications of the matrix W by vectors and $\mathcal{O}(N^2\log(N))$ other ops of this stage have been proved in [38]. Then, in $\mathcal{O}(N)$ ops, we compute the Sturm sequence of the N values of the determinants of all of the $k \times k$ northwestern (leading principal) submatrices of W for $k = 1, \dots, N$ and obtain the numbers N_+ and N_- of positive and negative eigenvalues of W from the Sturm sequence (cf., e.g., [3]). These two numbers immediately define the rank and the signature of W. ∎

Combining Proposition 3.11 with Theorems 3.13 and 3.14 gives us the next corollary.

COROLLARY 3.15. *For an $N \times N$ quasi-Toeplitz or quasi-Hankel matrix W, the estimates of Theorems* 3.13 *and* 3.14 *turn into $O(N^2\log(N))$ ops if the matrix has a maximal (c,d) support where $c+d=N$. They turn into $\mathcal{O}(N^2\log^2(N))$ ops if the matrix has a total degree (c,d) support where $c+d=O(N)$ and into $\mathcal{O}((\log^2(N)+\log(d))N^2+C_{W,N})$ otherwise, where d and $C_{W,N}$ are defined as in Proposition* 3.11 (a) *for $M=W$.*

REMARK 3.16. *Hereafter, we will refer to the matrices of case* (b) *in Proposition* 3.11 *as the matrices with support of the maximal degree (c,d) and to the matrices of case* (c) *as the ones with support of the total degree (c,d). Furthermore, stating our estimates for the arithmetic complexity of computations, we will assume that the input polynomials have the maximal degree (c,d) support. That is, we will rely on Theorem* 3.3 *and Proposition* 3.11 (b)*, and we will express the estimates in terms of the cardinality of the supports E and/or F or in terms of an upper bound D on the number of common roots of the input polynomials. The estimates can be easily extended to the other cases based on Theorem* 3.1 *or* 3.4 *and Proposition* 3.11 (a) *or* (c) *instead of Theorem* 3.3 *and Proposition* 3.11 (b)*. In the latter case (Theorem* 3.4 *and Proposition* 3.11 (c)*), the cost estimates increase by the factors* $\log(D)$, $\log(\lfloor E\rceil)$, *or* $\log(\lfloor F\rceil)$, *respectively. In case Theorems* 3.1 *and Proposition* 3.11 (a) *are used, the estimates are expressed in terms of the bounds $C_{PolMult}(G,H)$ or $C_{M,N}$ for appropriate sets G and H, matrix M, and integer N. The latter case covers sparse input polynomials for which the respective bounds $C_{PolMult}(G,H)$ and $C_{M,N}$ are smaller than for the general (or dense) input, although they are not expressed solely in terms of the cardinality D. (They also depend on the degree of the monomials or the cardinality of the supports of the input polynomial system.)*

4. Computation of a nondegenerate linear form. In this section, we will compute a nondegenerate linear form on $\mathcal{A}$ provided that we are given a basis $(\mathbf{x}^\alpha)_{\alpha\in E}$ of $\mathcal{A}$ and the normal form of the elements on the boundary of this basis. This is the case, for instance, when we have computed a Gröbner basis of our ideal I for any monomial ordering [9] or when we apply any other normal-form algorithm [28], [33].

DEFINITION 4.1.

- *Let $v_i = (\delta_{i,1}, \dots, \delta_{i,n}) \in \mathbb{N}^n$, where $\delta_{i,j}$ is the Kronecker symbol.*
- *For all $A \subset \mathbb{N}^n$, $\Omega(A) = \{\alpha \in \mathbb{N}^n : \alpha \in A \text{ or } \exists i \in \{1, \dots, n\}, \alpha - v_i \in A\}$.*
- *N_α for $\alpha \in \Omega(E)$ is the normal form of the monomial $\mathbf{x}^\alpha \bmod I$, i.e., the canonical representative of its class modulo the ideal I. $N_\alpha = \mathbf{x}^\alpha$ if $\alpha \in E$, and*

$$N_\alpha = \sum_{\beta\in E} n_{\alpha,\beta}\mathbf{x}^\beta$$

if $\alpha \in \Omega(E) - E$.

Our goal is to obtain the coefficients $\tau(\mathbf{x}^\alpha)$ for $\alpha \in E+E+E$, where $\tau \in \widehat{\mathcal{A}} = I^\perp$ is a generic linear form. We will compute them, by induction, under the following hypothesis.

HYPOTHESIS 4.2.

- *$(x^\alpha)_{\alpha\in E}$ is stable under derivation, that is, $\alpha = \alpha' + v_i \in E$ implies that $\alpha' \in E$.*
- *N_α, the normal form of $\mathbf{x}^\alpha$, is available for every $\alpha \in \Omega(E)$.*
- *The values $\tau_\alpha = \tau(x^\alpha)$ are available for all $\alpha \in E$, where τ is not degenerate $\in \widehat{\mathcal{A}} = I^\perp$.*

For the third part, we can remark that a random choice of $\tau(\mathbf{x}^\alpha)$ will imply with a high probability that τ does not degenerate. Our procedure is based on the following property.

PROPOSITION 4.3. *For each $\alpha \in \Omega(E)$, we have $\tau_\alpha = \tau(N_\alpha) = \sum_{\beta\in E} n_{\alpha,\beta}\tau_\beta$. This value can be computed by applying $\mathcal{O}(D)$ ops, where $D = \lfloor E \rceil$. More generally, for all $\gamma \in E$ we have the following inductive relation:*

$$\tau_{\alpha+\gamma} = \sum_{\beta\in E} n_{\alpha,\beta}\tau_{\beta+\gamma}.$$

Now assume that we have computed all of the values τ_β for $\beta \in \Omega(E)$, and let $\alpha = \alpha_0 + v_i \in \Omega(\Omega(E))$ with $\alpha_0 \in \Omega(E)$. Then

$$\tau(\mathbf{x}^\alpha) = \tau(x_i N_{\alpha_0}) = \sum_{\beta\in E} n_{\alpha_0,\beta}\tau\left(x_i\mathbf{x}^\beta\right).$$

We know all of the $n_{\alpha_0,\beta}$ and all of the $\tau\left(x_i\mathbf{x}^\beta\right)$ because $\beta + v_i \in \Omega(E)$. Therefore, we obtain $\tau_\alpha = \sum_{\beta\in E} n_{\alpha_0,\beta}\tau_{\beta+v_i}$ by computing a scalar product. Recursively, this leads us to the following inductive definition of the "levels" Ω_i.

DEFINITION 4.4. *Write $\Omega_0 = E$, $\Omega_1 = \Omega(E)$ and $\Omega_i = \Omega(\Omega_{i-1}) \cap (E + E + E)$, $i = 2, 3, \dots$, and write $h = \max\{|\alpha| : \alpha \in E\}$ so that $E + E + E = \Omega_{2h}$.*

PROPOSITION 4.5. *For every $\alpha \in \Omega_i$, there is $\alpha' \in \mathbb{N}^n$ and $\alpha_1 \in \Omega_1 - \Omega_0$ such that $\alpha = \alpha_1 + \alpha'$ with $|\alpha'| \le i - 1$ and for all $\beta \in E$ we have $\beta + \alpha' \in \Omega_{i-1}$.*

Proof. Assume that $i > 0$. Let $\alpha \in \Omega_i \subset E + E + E$. Then α can be decomposed as follows: $\alpha = \gamma_0 + \gamma_1 + \gamma_2$ with $\gamma_0, \gamma_1, \gamma_2 \in E$ and $|\gamma_1 + \gamma_2| = i$. As $i > 1$, there exists $\alpha' = \gamma_1 + \gamma_2 - v_j \in \mathbb{N}^n$, and because $(\mathbf{x}^\alpha)_{\alpha\in E}$ is stable by Hypothesis 4.2, we have

$\alpha' \in E + E$. It follows that $\alpha = \alpha_1 + \alpha'$, where $\alpha_1 = \gamma_0 + v_j \in \Omega_1$ and $|\alpha'| \leq i - 1$. Therefore, for all $\beta \in E$, $\beta + \alpha' \in \Omega_{i-1}$, which completes the proof. ∎

Assume now that we have already computed all of the values τ_β for $\beta \in \Omega_{i-1}$. Then, according to Proposition 4.5, for any $\alpha \in \Omega_i$, we have $\alpha = \alpha_1 + \alpha'$, with $\alpha_1 \in \Omega_1$ and $|\alpha'| \leq i - 1$. Thus, if $\alpha_1 \in \Omega_1 - \Omega_0$, we have

$$\tau(\mathbf{x}^\alpha) = \tau(\mathbf{x}^{\alpha_1}\, \mathbf{x}^{\alpha'}) = \sum_{\beta \in E} n_{\alpha_1,\beta} \tau(\mathbf{x}^{\beta+\alpha'})$$

with $\beta + \alpha' \in \Omega_{i-1}$; otherwise, if $\alpha_1 \in \Omega_0$, we have $\alpha = \alpha_1 + \alpha' \in \Omega_{i-1}$. In other words, we can compute by induction the values of τ on Ω_i from its values on Ω_{i-1}. This yields the following recursive algorithm for the computation of $\tau(\mathbf{x}^\alpha)$ with $\alpha \in E + E + E$.

ALGORITHM 4.6. *Compute the first coefficients of the series associated with a linear form τ of $I^\perp$ as follows:*

1. *For i from 1 to $2h$ do for each $\alpha = \alpha_0 + \alpha_1 \in \Omega_i$ with α_0 and α_1 as in Proposition 4.5 compute $\tau_\alpha = \sum_{\beta \in E} n_{\alpha_1,\beta} \tau_{\alpha_0+\beta}$*
   End for
2. *Compute and output the polynomial $S = \sum_{\alpha \in E+E+E} \tau_\alpha \mathtt{d}^\alpha$.*

PROPOSITION 4.7. *The arithmetic complexity of Algorithm 4.6 is $\mathcal{O}\left(3^n D^2\right)$.*

Proof. For each element $\alpha \in E + E + E$, we compute τ_α in $\mathcal{O}(D)$ arithmetic operations, and there are at most $\mathcal{O}(3^n D)$ elements in $E + E + E$, which gives us the claimed arithmetic complexity estimate. ∎

5. Arithmetic in the algebra $\mathcal{A}$. Our algorithms in the next sections perform computations in $\mathcal{A}$ efficiently based on the knowledge of a certain linear form on $\mathcal{A}$ (such as the one computed in the previous section), which induces a nondegenerate inner product. More precisely, we assume that the following items are available.

Basic set of items:

- *a linear form $\tau \in \widehat{\mathcal{A}} = I^\perp$, such that the bilinear form $\tau(a\, b)$ from $\mathcal{A} \times \mathcal{A}$ to $\mathbb{C}$ is nondegenerate,*
- *a monomial basis $(\mathbf{x}^\alpha)_{\alpha \in E}$ of $\mathcal{A}$,*
- *the coefficients $(\tau(\mathbf{x}^\alpha))_{\alpha \in F}$, where $F = E + E + E$.*

The number of elements in E is the dimension D of $\mathcal{A}$ over $\mathbb{C}$. We describe basic operations in the quotient ring $\mathcal{A}$ in terms of the following quasi-Hankel matrix.

DEFINITION 5.1. *For any Λ in $\widehat{\mathcal{A}}$ and for any subset F of $\mathbb{N}^n$, let $\mathtt{H}_\Lambda^F$ denote the quasi-Hankel matrix, $\mathtt{H}_\Lambda^F = (\Lambda(\mathbf{x}^{\alpha+\beta}))_{\alpha,\beta \in F}$.*

By default we will assume we are dealing with the maximal degree support whenever we state our arithmetic complexity estimates (see Remark 3.16).

PROPOSITION 5.2. *The matrix $\mathtt{H}_\Lambda^F$ can be multiplied by a vector by using $\mathcal{O}(3^n \lfloor F \rceil \log(3^n \lfloor F \rceil))$ ops.*

Proof. Apply Proposition 3.11 (b) to the (F, F) quasi-Hankel matrix $\mathtt{H}_\Lambda^F$, and observe that $\lfloor F + F + F \rceil = 3^n \lfloor F \rceil$. ∎

Combining Corollary 3.15 and Proposition 5.2 implies the following result.

PROPOSITION 5.3. *Check if the linear system $\mathtt{H}_\Lambda^F \mathbf{u} = \mathbf{v}$ has a unique solution, and, if so, computing the solution requires $\mathcal{O}(3^n \lfloor F \rceil^2 \log(3^n \lfloor F \rceil))$ ops. The same cost estimate applies to the computation of the rank of the matrix $\mathtt{H}_\Lambda^F$, which involves randomization with $\lceil F \rceil$ random parameters and has a failure probability of at most $(\lfloor F \rceil + 1) \lfloor F \rfloor / (2 \lfloor S \rceil)$ provided that the parameters have been sampled from a fixed finite set S.*

5.1. Dual basis. As τ defines a nondegenerate bilinear form, there exists a set of polynomials $(\mathbf{w}_\alpha)_{\alpha\in E}$ such that $\tau(\mathbf{x}^\alpha\,\mathbf{w}_\beta)=\delta_{\alpha,\beta}$, $\delta_{\alpha,\beta}$ being Kronecker's symbol, $\delta_{\alpha,\alpha}=1$, and $\delta_{\alpha,\beta}=0$ if $\alpha\neq\beta$. The set $(\mathbf{w}_\alpha)_{\alpha\in E}$ is called the *dual basis* of $(\mathbf{x}^\alpha)_{\alpha\in E}$ for τ.

PROPOSITION 5.4 (projection formula). *For any $p\in R$, we have*

$$p\equiv\sum_{\alpha\in E}\tau(p\,\mathbf{w}_\alpha)\mathbf{x}^\alpha\equiv\sum_{\alpha\in E}\tau(p\,\mathbf{x}^\alpha)\mathbf{w}_\alpha. \tag{4}$$

Proof. See [7], [11]. □

DEFINITION 5.5. *For any $p\in\mathcal{A}$, denote by $[p]_{\mathbf{x}}$ and $[p]_{\mathbf{w}}$ the coordinate vectors of p in the bases $(\mathbf{x}^\alpha)_{\alpha\in E}$ and $(\mathbf{w}_\alpha)_{\alpha\in E}$, respectively.*

Let $\mathbf{w}_\alpha=\sum_{\beta\in E}w_{\beta,\alpha}\,\mathbf{x}^\beta$, and let $\mathtt{W}_\tau=(w_{\alpha,\beta})_{\alpha,\beta\in E}$ be the coefficient matrix. By the definition of the dual basis,

$$\tau(\mathbf{w}_\alpha\,\mathbf{x}^\gamma)=\sum_{\beta\in E}w_{\alpha,\beta}\,\tau(\mathbf{x}^{\beta+\gamma}) \tag{5}$$

is 1 if $\alpha=\gamma$ and 0 elsewhere. In terms of matrices, (5) implies that

$$\mathtt{H}_\tau\,\mathtt{W}_\tau=\mathbb{I}_D, \tag{6}$$

where $\mathtt{H}_\tau=\mathtt{H}_\tau^E=(\tau(\mathbf{x}^{\beta+\gamma}))_{\beta,\gamma\in E}$. From the definition of $\mathtt{W}_\tau$ and (6), we deduce that

$$[p]_{\mathbf{x}}=\mathtt{W}_\tau\,[p]_{\mathbf{w}},\ [p]_{\mathbf{w}}=\mathtt{H}_\tau\,[p]_{\mathbf{x}}. \tag{7}$$

The next result follows from Proposition 5.3.

PROPOSITION 5.6. *For any $p\in\mathcal{A}$, the coordinates $[p]_{\mathbf{x}}$ of p in the monomial basis can be computed from its coordinates $[p]_{\mathbf{w}}$ in the dual basis by using $\mathcal{O}(3^nD^2\,\log(3^nD))$ ops.*

5.2. Product in $\mathcal{A}$. We apply projection formula (4) and, for any $f\in R$, deduce that $f\equiv\sum_{\alpha\in E}\tau(f\,\mathbf{x}^\alpha)\,\mathbf{w}_\alpha=\sum_{\alpha\in E}f\star\tau(\mathbf{x}^\alpha)\,\mathbf{w}_\alpha$ in $\mathcal{A}$. Furthermore, by expressing the linear form $f\star\tau$ as a formal power series, we obtain $f\star\tau=\sum_{\alpha\in\mathbb{N}^n}\,f\star\tau(\mathbf{x}^\alpha)\,\mathbf{d}^\alpha$ so that the coefficients of $(\mathbf{d}^\alpha)_{\alpha\in E}$ in the expansion of $f\star\tau$ are the coefficients $[f]_{\mathbf{w}}$ of f in the dual basis $(\mathbf{w}_\alpha)_{\alpha\in E}$.

Similarly, for any $f,g\in\mathcal{A}$, the coefficients of $(\mathbf{d}^\alpha)_{\alpha\in E}$ in $fg\star\tau$ are the coefficients $[fg]_{\mathbf{w}}$ of $f\,g$ in the dual basis $(\mathbf{w}_\alpha)_{\alpha\in E}$. This leads to the following algorithm for computing the product in $\mathcal{A}$.

ALGORITHM 5.7. *For any pair $f,g\in\langle\mathbf{x}^\alpha\rangle_{\alpha\in E}$, compute the product fg in the basis $\langle\mathbf{x}^\alpha\rangle_{\alpha\in E}$ of $\mathcal{A}$ as follows:*

1. *Compute the coefficients of $(\mathbf{d}^\alpha)_{\alpha\in E}$ in the product $f\,g\star\tau$.*
2. *Obtain the coefficients $[f\,g]_{\mathbf{w}}$ from the first coefficients of $fg\star\tau$.*
3. *Solve in $\mathbf{u}$ the linear system $[f\,g]_{\mathbf{w}}=\mathtt{H}_\tau\,\mathbf{u}$.*

Output the vector $\mathbf{u}$, which is the coordinate vector $[f\,g]_{\mathbf{x}}$ of $f\,g$ in the monomial basis of $\mathcal{A}$.

PROPOSITION 5.8. *The product $f\,g$ can be computed in $\mathcal{O}(3^nD^2\log(3^nD))$ ops.*

Proof. $f\,g\star\tau$ is the product of polynomials with supports in $-E$ or $E+E+E$. Such a product can be computed in $\mathcal{O}(3^nD\log^2(3^nD))$ ops (see Proposition 3.11 and Remark 3.16 and observe that $\lfloor E+E+E\rceil=\mathcal{O}\,(3^n\lfloor E\rceil)$). The complexity of the third step is bounded according to Proposition 5.3 (with $F=E$). □

5.3. Inversion in $\mathcal{A}$. The projection formula of Proposition 5.4 implies that $f\,\mathbf{x}^\alpha = \sum_{\beta\in E} f \star \tau(\mathbf{x}^{\alpha+\beta})\,\mathbf{w}_\beta$, which means that $[f\,\mathbf{x}^\alpha]_{\mathbf{w}}$ is the coordinate vector $[f \star \tau(\mathbf{x}^{\alpha+\beta})]_{\beta\in E}$, that is, the column of the matrix $\mathtt{H}_{f\star\tau}$ indexed by α. In other words, $[f\,\mathbf{x}^\alpha]_{\mathbf{w}} = \mathtt{H}_{f\star\tau}\,[\mathbf{x}^\alpha]_{\mathbf{x}}$. By linearity, for any $g \in \mathcal{A}$, we have

$$[f\,g]_{\mathbf{w}} = \mathtt{H}_{f\star\tau}[g]_{\mathbf{x}} = \mathtt{H}_\tau\,[f\,g]_{\mathbf{x}},$$

according to (7). Thus, if $fg = 1$, that is, if $g = f^{-1}$, we have $\mathtt{H}_{f\star\tau}[g]_{\mathbf{x}} = \mathtt{H}_\tau\,[1]_{\mathbf{x}}$. This leads to the following algorithm for computing the inverses (reciprocals) in $\mathcal{A}$.

ALGORITHM 5.9. *For any $f \in \langle \mathbf{x}^\alpha\rangle_{\alpha\in E}$, verify whether there exists the inverse (reciprocal) of $f \in \mathcal{A}$, and, if so, compute it.*

1. *Compute* $\mathbf{v} = \mathtt{H}_\tau\,[1]_{\mathbf{x}}$.
2. *Solve in* $\mathbf{u}$ *the linear system* $\mathtt{H}_{f\star\tau}\mathbf{u} = \mathbf{v}$ *or output* FAILURE *if the matrix* $\mathtt{H}_\tau$ *is not invertible.*

Output the vector $\mathbf{u}$*, which is the coordinate vector* $[f^{-1}]_{\mathbf{x}}$ *of* f^{-1} *in the monomial basis of* $\mathcal{A}$.

By combining Propositions 5.2 and 5.3 and Remark 3.16, we obtain the following proposition.

PROPOSITION 5.10. *The inverse (reciprocal)* f^{-1} *of an element* f *of* $\mathcal{A}$ *can be computed by using* $\mathcal{O}(3^n D^2 \log(3^n D))$ *ops.*

6. Iterative methods. Our algorithms for the root approximation will essentially amount to computing nontrivial idempotents in the quotient algebra $\mathcal{A}$ by iterative processes with the subsequent simple recovery of the roots from the idempotents. The algorithms work in $\mathbb{C}^D$, and we will write $\mathbf{i} = \sqrt{-1}$. More rudimentary univariate versions of such algorithms were studied in [6]. We will use the basic operations in the quotient algebra $\mathcal{A}$ in order to devise two iterative methods, which will converge to nontrivial idempotents. We will first consider an iteration associated to a slight modification of the so-called *Joukovski* map (see [20], [6]): $z \mapsto \frac{1}{2}(z + \frac{1}{z})$ and its variant $z \mapsto \frac{1}{2}(z - \frac{1}{z})$. The two attractive fixed points of this map are 1 and -1; for its variant, they turn into $\mathbf{i}$ and $-\mathbf{i}$.

ALGORITHM 6.1. *Sign iteration. Choose* $u_0 = h \in \langle \mathbf{x}^\alpha\rangle_{\alpha\in E}$*, and recursively compute* $u_{k+1} \equiv \frac{1}{2}(u_k - \frac{1}{u_k}) \in \mathcal{A},\ k = 0, 1, \dots$.

By applying Proposition 5.10 and Remark 3.16, we obtain the following result.

PROPOSITION 6.2. *Each iteration of Algorithm* 6.1 *requires* $\mathcal{O}(3^n D^2 \log(3^n D))$ *ops.*

Proof. Apply Proposition 5.3 and Remark 3.16 to estimate the arithmetic cost of the computation of the inverse (reciprocal) of an element of $\mathcal{A}$. To yield the claimed cost bound of Proposition 6.2, it remains to compute a linear combination of u_n and u_n^{-1} in $\mathcal{O}(D)$ ops by direct operations on vectors of size D. ∎

Hereafter, $\Re(h)$ and $\Im(h)$ denote the real and the imaginary parts of a complex number h, respectively. Recall that we write ζ to denote the common roots $\zeta \in \mathcal{Z}(I)$ of given polynomials $f_1, \dots, f_m$.

REMARK 6.3. *In Proposition* 6.4*, we will assume that* $J(h(\zeta)) \neq 0$ *for all* $\zeta \in \mathcal{Z}(I)$ *and, in Proposition* 6.6*, that* $|h(\zeta)|$ *is minimized for a unique root* $\zeta \in \mathcal{Z}(I)$*. These assumptions are satisfied for a generic system of polynomials or a generic polynomial* h.

PROPOSITION 6.4. *The sequence* $(u_0, u_1, \dots)$ *of Algorithm* 6.1 *converges quadratically to* $\sigma = \sum_{\Im(h(\zeta))>0} \mathbf{e}_\zeta - \sum_{\Im(h(\zeta))<0} \mathbf{e}_\zeta$*, and we have*

$$\|u_n - \sigma\| \le K \times \rho^{2^n}$$

(for some constant K), where

$$\rho^+ = \max_{\Im(h(\zeta))>0, \zeta\in\mathcal{Z}(I)} \left|\frac{h(\zeta)-\mathbf{i}}{h(\zeta)+\mathbf{i}}\right|,$$

$$\rho^- = \max_{\Im(h(\zeta))<0, \zeta\in\mathcal{Z}(I)} \left|\frac{h(\zeta)+\mathbf{i}}{h(\zeta)-\mathbf{i}}\right|,$$

$\mathbf{i}=\sqrt{-1}$, *and* $\rho=\max\{\rho^+,\rho^-\}$.

Proof. Apply the classical convergence analysis of the Joukovski map (see [20]) to the matrices of multiplication by u_n in $\mathcal{A}$, whose eigenvalues are $\{u_n(\zeta), \zeta\in\mathcal{Z}(I)\}$. □

Let

$$\mathbf{e}^+ = \sum_{\Im(h(\zeta))>0} \mathbf{e}_\zeta = \frac{1}{2}(1+\sigma), \quad \mathbf{e}^- = \sum_{\Im(h(\zeta))\le 0} \mathbf{e}_\zeta = \frac{1}{2}(1-\sigma)$$

denote the two sums of the idempotents associated to the roots $\zeta\in\mathcal{Z}$ such that $\Im(h(\zeta))>0$ and $\Im(h(\zeta))<0$, respectively.

If $h(\mathbf{x})$ is a linear function in $\mathbf{x}$, then each of the idempotents $\mathbf{e}^+$ and $\mathbf{e}^-$ is associated with all of the roots lying in a fixed half-space of $\mathbb{C}^n$ defined by the inequalities $\Im(h(\zeta))>0$ or $\Im(h(\zeta))<0$. Conversely, an appropriate linear function $h(\mathbf{x})$ defines the idempotents $\mathbf{e}^+$ and $\mathbf{e}^-$ associated with any fixed half-space of $\mathbb{C}^n$. Furthermore, for any fixed polytope in $\mathbb{C}^n$ defined as the intersection of half-spaces, we may compute the family of the associated idempotents whose product will be associated with the polytope. In particular, any bounded box is the intersection of $4n$ half-spaces, and the associated idempotent can be computed in $4n$ applications of Algorithm 6.1. Let us specify the case in which the polytope is the almost flat unbounded box approximating the real manifold $R^n=\{\mathbf{x}:\Im(x_i)=0, i=1,\dots,n\}$. In this case, the choices of $h=x_i-\epsilon$ and $h=x_i+\epsilon$ allow us to approximate the two idempotents

$$\mathbf{e}^-_{i,\epsilon} = \sum_{\Im(\zeta_i)<\epsilon} \mathbf{e}_\zeta, \quad \mathbf{e}^+_{i,\epsilon} = \sum_{\Im(\zeta_i)>-\epsilon} \mathbf{e}_\zeta.$$

Their product can be computed in $\mathcal{O}(3^nD^2\log(3^nD))$ ops to yield $\mathbf{r}_{i,\epsilon}=\sum_{|\Im(\zeta_i)|<\epsilon}\mathbf{e}_\zeta$, and the product $\mathbf{r}_\epsilon\equiv\mathbf{r}_{1,\epsilon}\cdots\mathbf{r}_{n,\epsilon}$ can be computed in $\mathcal{O}(3^nD^2\log(3^nD))$ ops to yield the sum of the fundamental idempotents whose associated roots of the polynomial system are nearly real.

ALGORITHM 6.5. *Computing the sum of the fundamental (nearly real) idempotents.*

- *For i from 1 to n do*
  $u_0=x_i\pm\epsilon$; $u_1:\equiv\frac{1}{2}(u_0-\frac{1}{u_0})$ in $\mathcal{A}$; $k:=1$;
  while $\|u_k-u_{k-1}\|<2^{-b}$ do $\{\ u_{k+1}:=\frac{1}{2}(u_k-\frac{1}{u_k})$; $k:=k+1\ \}$
  Compute $\mathbf{e}^\pm_{i,e}$ and $\mathbf{r}_{i,\epsilon}$.
- *Compute and output the product $\mathbf{r}_\epsilon\equiv\mathbf{r}_{1,\epsilon}\cdots\mathbf{r}_{n,\epsilon}$ in $\mathcal{A}$.*

According to Propositions 6.2 and 6.4 and Remark 3.16, we have the following proposition.

PROPOSITION 6.6. *An approximation of $\mathbf{r}_\epsilon$ (within the error bound $\epsilon=2^{-b}$) can be computed in $\mathcal{O}(\mu\,3^nD^2\ \log(3^nD))$ ops, where*

$$\mu=\mu(b,\rho)=\log|b/\log(\rho)| \tag{8}$$

and

$$\rho = \max_i\{ \quad \max_{\Im(\zeta_i)>0,\zeta\in\mathcal{Z}(I)} |\tfrac{\zeta_i-\mathbf{i}}{\zeta_i+\mathbf{i}}|, \quad \max_{\Im(\zeta_i)<0,\zeta\in\mathcal{Z}(I)} |\tfrac{\zeta_i+\mathbf{i}}{\zeta_i-\mathbf{i}}| \quad \}. \tag{9}$$

The second iterative method is the quadratic power method.

ALGORITHM 6.7. *Quadratic power iteration. Choose* $u_0 = h \in \langle \mathbf{x}^\alpha \rangle_{\alpha\in E}$, *and recursively compute* $u_{n+1} \equiv u_n^2 \in \mathcal{A}$, $n = 0, 1, \ldots$.

Each step of this iteration requires at most $\mathcal{O}\left(3^n D^2 \log\left(3^n D\right)\right)$ ops, and we have the following property.

PROPOSITION 6.8. *An approximation (within the error bound* $\epsilon = 2^{-b}$*) of the idempotent* $\mathbf{e}_\zeta$ *such that a unique simple root* ζ *minimizes* $|h|$ *on* $\mathcal{Z}(I)$ *can be computed in* $\mathcal{O}\left(\nu 3^n D^2 \log\left(3^n D\right)\right)$ *ops, where*

$$\nu = \nu\left(b, \gamma\right) = \log\left(b/|\log\left(\gamma\right)|\right), \tag{10}$$

$$\gamma = \left|\frac{h(\zeta)}{h(\zeta')}\right|, \tag{11}$$

and $|h(\zeta')|$ *is the second smallest value of* $|h|$ *over* $\mathcal{Z}(I)$.

Proof. We rely on the convergence analysis of the quadratic power method applied to the matrices of multiplication by u_n in $\mathcal{A}$, whose eigenvalues are $\{u_n(\zeta), \zeta \in \mathcal{Z}(I)\}$. ∎

7. Counting and approximating the roots and the real roots. In this section, we will apply the techniques and algorithms of the previous sections to the problems of counting and approximation of the roots of the system $\mathbf{p} = \mathbf{0}$.

In the algorithms for counting roots, we will use the randomization required to apply Theorem 3.13. The resulting randomized algorithms and the computational complexity estimates for counting (excluding the preprocessing stage of subsection 7.5) will apply to any 0-dimensional polynomial system.

In the approximation algorithms, we do not need randomization except for the ensurance of the assumption of Propositions 6.4 (cf. Remark 6.3), but the estimates for the computational cost depend on the parameters ρ and γ of the two latter propositions (cf. (8), (11)) and remain meaningful unless these parameters are extremely close to 1.

7.1. Counting the roots and the real roots.

THEOREM 7.1 (see [29]). *The number of the roots (resp., real roots) of the system* $\mathbf{p} = \mathbf{0}$ *is given by the rank (resp., the signature) of the quasi-Hankel matrix* H_τ^E.

Theorem 7.1, Corollary 3.15, and Remark 3.16 together imply the following result.

COROLLARY 7.2. *The numbers of the roots and of the real roots of the polynomial system* $\mathbf{p} = \mathbf{0}$ *can be computed by a randomized algorithm that generates* D *random parameters and, in addition, performs* $O(3^n D^2 \log(3^n D))$ *ops. If the random parameters are sampled from a fixed finite set* S, *then the algorithm may fail with a probability at most* $\left(3^n D + 1\right) 3^n D / \left(2\lfloor S \rceil\right)$.

7.2. Approximation of a root. Application of Algorithm 6.7 in $\mathcal{A}$ yields the following theorem.

THEOREM 7.3. *The idempotent corresponding to a root* ζ *that maximizes the absolute values* $|h(\zeta)|$ *of a fixed polynomial* $h(\mathbf{x})$ *can be approximated (within an error bound* $\epsilon = 2^{-b}$*) by using* $O(3^n D^2 \nu \log(3^n D))$ *ops, where* ν *is defined in* (10) *and* (11).

The latter cost bound dominates the cost of the subsequent transition from the idempotent to a root.

THEOREM 7.4. *The n coordinates of a simple root ζ can be determined from the idempotent $\mathbf{e}_\zeta$ in $\mathcal{O}(3^n D^2 \log(3^n D))$ ops. This bound increases by the factor of n if the root is multiple.*

Proof. We compute $J\mathbf{e}_\zeta$ in $\mathcal{A}$ (where J is the Jacobian of the n equations) by Algorithm 5.7. According to [29], [32], in the case of a simple root, we have

$$\mathtt{H}_\tau^E\,[J\,\mathbf{e}_\zeta]_{\mathbf{x}} = \lambda\,[\zeta^\alpha]_{\alpha\in E}, \quad \lambda \in \mathbb{C}.$$

This vector is computed at the arithmetic cost within the complexity bound of Proposition 5.2 (cf. [32]), and this immediately gives us the coordinates of the root ζ if $\mathbf{x}^E$ contains $1, x_1, \dots, x_n$, which is generically the case. If the root is not simple, then, according to the relation

$$x_i\,J\,\mathbf{e}_\zeta \equiv \zeta_i\,J\,\mathbf{e}_\zeta$$

(see [29], [32], [11]), we recover the coordinates of ζ by computing $n+1$ products in $\mathcal{A}$ (by Algorithm 5.7). □

7.3. Approximation of a selected root. In view of Theorem 7.4, it is sufficient to approximate the idempotents associated to the roots.

Suppose that we seek a root of the system $\mathbf{p} = \mathbf{0}$ whose coordinate x_1 is the closest to a given value $u \in \mathbb{C}$. Let us assume that u is not a projection of any root of the system $\mathbf{p} = \mathbf{0}$ so that $x_1 - u$ has the inverse (reciprocal) in $\mathcal{A}$. Let $h(\mathbf{x})$ denote such an inverse (reciprocal). We have $h\,(\mathbf{x})(x_1 - u) \equiv 1$ and $h(\zeta) = \frac{1}{\zeta_1 - u}$. Therefore, a root whose coordinate x_1 is the closest to u_1 is a root for which $|h(\zeta)|$ is the largest. Consequently, iterative squaring of $h = h(\mathbf{x})$ shall converge to this root.

The polynomial h can be computed by using $O(3^n D^2 \nu \log(3^n D))$ ops for ν of (10) and (11) (see [32, section 3.3.4]).

One may compute several roots of the polynomial system by applying the latter computation (successively or concurrently) to several initial values u.

7.4. Counting nearly real roots and the roots in a polytope. As long as we have (a close approximation to) the idempotent $\mathbf{r}$ associated with a fixed polytope, we may restrict our counting and approximation algorithms to such a polytope simply by moving from the basic nondegenerate linear form τ to the form $\mathbf{r} \star \tau$ (by using $O(3^n D^2 \log(3^n D))$ ops). Let us specify this in the case in which the polytope is the nearly flat box approximating the real space $\mathbb{R}^n$ (cf. Algorithm 6.5 and Proposition 6.6).

Let $\mathcal{A}_\epsilon^{\mathbb{R}} = \mathbf{r}_\epsilon \mathcal{A}$ denote the subalgebra of $\mathcal{A}$ corresponding to the (nearly) real idempotents for a fixed $\epsilon = 2^{-b}$.

We may restrict our computation on $\mathcal{A}_\epsilon^{\mathbb{R}}$ by computing the linear form $\tau' = \mathbf{r}_\epsilon \star \tau$ (in $\mathcal{O}(3^n D^2 \log(D))$ ops, according to Proposition 3.12), and we have the following properties.

PROPOSITION 7.5.

- *The linear form $\tau' = \mathbf{r}_\epsilon \star \tau$ defines a nondegenerate inner product on $\mathcal{A}_\epsilon^{\mathbb{R}}$.*
- *The number of nearly real roots (counted with their multiplicities) is the rank of the matrix $H^E_{\mathbf{r}_\epsilon \star \tau} = (\mathbf{r}_\epsilon \star \tau(\mathbf{x}^{\beta+\gamma}))_{\beta,\gamma \in F}$.*
- *Let E' be a subset of E such that the submatrix $H^{E'}_{\tau'}$ is of the maximal rank. Then E' is a basis of $\mathcal{A}_\epsilon$.*

Proof. See [32]. □

We thus require an algorithm for computing the rank of $H_{\tau'}^{E}$ (see [35] on fast computation of the rank). Assuming (8) and (9), we deduce the following result from Theorem 3.14.

PROPOSITION 7.6. *The number of all nearly real roots can be computed by using* $\mathcal{O}(\mu\, 3^n\, D^2 \log(3^n D))$ *ops (for* μ *of* (8) *and* (9)*).*

7.5. Approximation of nearly real roots and the roots in a box.. To compute a nearly real root as well as a root lying in a fixed box in $\mathbb{C}^n$ maximizing a given function $|h|$, we may apply Algorithm 6.7 in $\mathcal{A}$ (or $\mathcal{A}_{\epsilon}^{\mathbb{R}}$) and Proposition 6.8 and obtain the following theorem.

THEOREM 7.7. *A nearly real root (as well as a root lying in a fixed box) that maximizes a function* $|h|$ *can be computed (up to an error* $\epsilon = 2^{-b}$*) by using* $\mathcal{O}((\mu+\nu)\ 3^n D^2 \log(3^n D))$ *ops for* μ *and* ν *of* (8)–(11)*.*

This process can be extended to compute the other roots via deflation. That is, we replace $\mathbf{r}_\epsilon$ by $\mathbf{r}'_\epsilon = \mathbf{r}_\epsilon - \mathbf{e}_\zeta$, compute $\tau'' = \mathbf{r}'_\epsilon \star \tau$, and apply the same iteration to compute the next (real) root, where $|h|$ takes on its second smallest value over $\mathcal{Z}(I)$. We can also restrict our computation to a fixed box by using the algorithm of subsection 7.4 to compute the sum of the idempotents corresponding to the roots lying inside the box. The complexity of each step is bounded in Theorem 7.7, leading to the following result for δ (real) roots in a given box.

THEOREM 7.8. *The* δ *(real) roots* ζ *lying in a given box can be computed (up to an error* $\epsilon = 2^{-b}$*) by using* $\mathcal{O}((\mu+\nu)\, n 3^n \delta\, D^2\, \log(D)\log(b))$ *ops for* μ *and* ν *of* (8)–(11)*.*

REFERENCES

[1] W. AUZINGER AND H. J. STETTER, *An elimination algorithm for the computation of all zeros of a system of multivariate polynomial equations*, in Numerical Mathematics (Singapore, 1988), Internat. Schriftenreihe Numer. Math. 86, Birkhäuser, Basel, 1988, pp. 11–30.

[2] D. BINI AND V. Y. PAN, *Polynomial and matrix computations, Vol.* 1: *Fundamental Algorithms*, Birkhäuser Boston, Boston, 1994.

[3] D. BINI AND V. Y. PAN, *Computing matrix eigenvalues and polynomial zeros where the output is real*, SIAM J. Comput., 27 (1998), pp. 1099–1115.

[4] J. CANNY, *Generalised characteristic polynomials*, J. Symbolic Comput., 9 (1990), pp. 241–250.

[5] J. CANNY AND I. EMIRIS, *An efficient algorithm for the sparse mixed resultant*, in Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (San Juan, PR, 1993), Lecture Notes in Comput. Sci. 673, G. Cohen, T. Mora, and O. Moreno, eds., Springer-Verlag, Berlin, 1993, pp. 89–104.

[6] J. P. CARDINAL, *On two iterative methods for approximating the roots of a polynomial*, in The Mathematics of Numerical Analysis (Park City, UT, 1995), Lectures in Appl. Math. 32, J. Renegar, M. Shub, and S. Smale, eds., AMS, Providence, RI, 1996, pp. 165–188.

[7] J. P. CARDINAL AND B. MOURRAIN, *Algebraic approach of residues and applications*, in The Mathematics of Numerical Analysis (Park City, UT, 1995), Lectures in Appl. Math. 32, J. Renegar, M. Shub, and S. Smale, eds., AMS, Providence, RI, 1996, pp. 189–210.

[8] A. L. CHISTOV AND D. Y. GRIGORIEV, *Complexity of Quantifier Elimination in the Theory of Algebraically Closed Fields*, Lecture Notes in Comput. Sci. 176, Springer-Verlag, New York, 1984.

[9] D. COX, J. LITTLE, AND D. O'SHEA, *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Undergrad. Texts Math., Springer-Verlag, New York, 1992.

[10] D. COX, J. LITTLE, AND D. O'SHEA, *Using Algebraic Geometry*, Undergrad. Texts Math., Springer-Verlag, New York, 1998.

[11] M. ELKADI AND B. MOURRAIN, *Approche effective des résidus algébriques*, Rapport de recherche 2884, INRIA, Sophia-Antipolis, France, 1996.

[12] M. Elkadi and B. Mourrain, *Some Applications of Bezoutians in Effective Algebraic Geometry*, Rapport de recherche 3572, INRIA, Sophia-Antipolis, France, 1998.

[13] M. Elkadi and B. Mourrain, *A new algorithm for the geometric decomposition of a variety*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation, S. Dooley, ed., ACM, New York, 1999, pp. 9–16.

[14] M. Elkadi and B. Mourrain, *Algorithms for residues and Lojasiewicz exponents*, J. Pure Appl. Algebra, 153 (2000), pp. 27–44.

[15] I. Z. Emiris and V. Y. Pan, *The structure of sparse resultant matrices*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation, ACM, New York, 1997, pp. 189–196.

[16] I. Z. Emiris and A. Rege, *Monomial bases and polynomial system solving*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation, ACM, New York, 1994, pp. 114–122.

[17] J. C. Faugère, *A new efficient algorithm for computing Gröbner basis (F4)*, J. Pure Appl. Algebra, 139 (1999), pp. 61–88.

[18] M. Giusti and J. Heintz, *La détermination des points isolés et de la dimension d'une variété algébrique peut se faire en temps polynomial*, in Proceedings of the International Meeting on Commutative Algebra, Sympos. Math. 34, Cambridge University Press, Cambridge, UK, 1991, pp. 216–255.

[19] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., John Hopkins University Press, Baltimore, MD, 1996.

[20] P. Henrici, *Applied and Computational Complex Analysis. Volume* I, Wiley, New York, 1988.

[21] D. Kapur and Y. N. Lakshman, *Elimination methods: An introduction*, in Symbolic and Numerical Computation for Artificial Intelligence, B. Donald, D. Kapur, and J. Mundy, eds., Academic Press, New York, 1992, pp. 45–89.

[22] Y. N. Lakshman and D. Lazard, *On the complexity of zero-dimensional algebraic systems*, in Effective Methods in Algebraic Geometry, Progr. Math. 94, Birkhäuser Boston, Boston, 1991, pp. 217–225.

[23] F. S. Macaulay, *The Algebraic Theory of Modular Systems*, Cambridge University Press, Cambridge, UK, 1916.

[24] J. M. Rojas, *On the average number of real roots of certain random sparse polynomial systems*, in The Mathematics of Numerical Analysis (Park City, UT, 1995), Lectures in Appl. Math. 32, J. Renegar, M. Shub, and S. Smale, eds., AMS, Providence, RI, 1996, pp. 689–699.

[25] D. Manocha, *Solving polynomial systems using matrix computations*, in Advances in Computational Mathematics, Ser. Approx. Decompos. 4, World Scientific Publishing, River Edge, NJ, 1994, pp. 99–129.

[26] B. Mourrain, *Isolated points, duality and residues*, J. Pure Appl. Algebra, 117/118 (1996), pp. 469–493.

[27] B. Mourrain, *Computing isolated polynomial roots by matrix methods*, J. Symbolic Comput., 26 (1998), pp. 715–738.

[28] B. Mourrain, *A new criterion for normal form algorithms*, in Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, Lecture Notes in Comput. Sci. 1719, M. Fossorier, H. Imai, S. Lin, and A. Poli, eds., Springer-Verlag, Berlin, 1999, pp. 430–443.

[29] B. Mourrain and V. Y. Pan, *Multidimensional structured matrices and polynomial systems*, Calcolo, 33 (1997), pp. 389–401.

[30] B. Mourrain and V. Y. Pan, *Solving special polynomial systems by using structured matrices and algebraic residues*, in Foundations of Computational Mathematics (Rio de Janeiro), F. Cucker and M. Shub, eds., Springer-Verlag, New York, 1997, pp. 287–304.

[31] B. Mourrain and V. Y. Pan, *Asymptotic acceleration of solving multivariate polynomial systems of equations*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 488–496.

[32] B. Mourrain and V. Y. Pan, *Multivariate polynomials, duality and structured matrices*, J. Complexity, 16 (2000), pp. 110–180.

[33] B. Mourrain and P. Trébuchet, *Solving projective complete intersection faster*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation, ACM, New York, 2000, pp. 231–238.

[34] V. Y. Pan, *Optimal (up to polylog factors) sequential and parallel algorithms for approximating complex polynomial zeros*, in Proceedings of the 27th Annual Symposium on Theory of Computing, ACM, New York, 1995, pp. 741–750.

[35] V. Y. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*, Birkhäuser Boston, Springer-Verlag, New York, 2001.

[36] V. Y. Pan, *Optimal and nearly optimal algorithms for approximating complex polynomial zeros*, Comput. Math. Appl., 31 (1996), pp. 97–138.

[37] V. Y. Pan, *Solving a polynomial equation: Some history and recent progress*, SIAM Rev., 39 (1997), pp. 187–220.

[38] V. Y. Pan and Z. Chen, *The complexity of matrix eigenproblem*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 507–516.

[39] J. Renegar, *On the worst-case complexity of approximating zeros of polynomials*, J. Complexity, 3 (1987), pp. 90–113.

[40] J. Renegar, *On the worst-case arithmetic complexity of approximating zeros of systems of polynomials*, SIAM J. Comput., 18 (1989), pp. 350–370.

[41] M. Shub and S. Smale, *On the complexity of Bezout's theorem* I—*geometric aspects*, J. Amer. Math. Soc., 6 (1993), pp. 459–501.

[42] H. J. Stetter, *Eigenproblems are at the heart of polynomial system solving*, SIGSAM Bulletin, 30 (1996), pp. 22–25.

[43] H. J. Stetter, *Analysis of zero clusters in multivariate polynomial systems*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation, ACM, New York, 1996, pp. 127–135.

[44] E. E. Tyrtyshnikov, *A unifying approach to some old and new theorems on distribution and clustering*, Linear Algebra Appl., 232 (1996), pp. 1–43.

[45] B. L. van der Waerden, *Modern Algebra, Volume* II, Frederick Ungar Publishing, New York, 1948.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 455–469

NEW BOUNDS FOR VARIABLE-SIZED ONLINE BIN PACKING*

STEVEN S. SEIDEN†, ROB VAN STEE‡, AND LEAH EPSTEIN§

The remaining authors would like to dedicate this paper to the memory of our friend Steve Seiden, who was killed in an accident on June 11, 2002

Abstract. In the variable-sized online bin packing problem, one has to assign items to bins one by one. The bins are drawn from some fixed set of sizes, and the goal is to minimize the sum of the sizes of the bins used. We present new algorithms for this problem and show upper bounds for them which improve on the best previous upper bounds. We also show the first general lower bounds for this problem. The case in which bins of two sizes, 1 and $\alpha \in (0, 1)$, are used is studied in detail. This investigation leads us to the discovery of several interesting fractal-like curves.

Key words. bin packing, online algorithms

AMS subject classifications. 68Q25, 68W25, 68W40

PII. S0097539702412908

1. Introduction.

In this paper, we investigate the bin packing problem, one of the oldest and most thoroughly studied problems in computer science [3, 5]. In particular, we investigate a natural generalization of the classical online bin packing problem known as online variable-sized bin packing. We show improved upper bounds and the first lower bounds for this problem and in the process encounter several strange fractal-like curves.

Problem definition. In the *classical bin packing* problem, we receive a sequence σ of *pieces* $p_1, p_2, \ldots, p_N$. Each piece has a fixed *size* in $(0, 1]$. In a slight abuse of notation, we use p_i to indicate both the ith piece and its size. We have an infinite number of *bins* each with *capacity* 1. Each piece must be assigned to a bin. Further, the sum of the sizes of the pieces assigned to any bin may not exceed its capacity. A bin is *empty* if no piece is assigned to it; otherwise, it is *used*. The goal is to minimize the number of bins used.

The *variable-sized bin packing* problem differs from the classical one in that the bins do not all have the same capacity. There are an infinite number of bins of each capacity $\alpha_1 < \alpha_2 < \cdots < \alpha_m = 1$. The goal now is to minimize the sum of the capacities of the bins used.

In the *online* versions of these problems, each piece must be assigned in turn, without knowledge of the next pieces. Since it is impossible in general to produce the best possible solution when computation occurs online, we consider approximation

*Received by the editors August 14, 2002; accepted for publication (in revised form) October 1, 2002; published electronically February 4, 2003. A preliminary version of this paper appeared in *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, 2002, pp. 306–317.

http://www.siam.org/journals/sicomp/32-2/41290.html

†The author is deceased. Former address: Department of Computer Science, 298 Coates Hall, Louisiana State University, Baton Rouge, LA 70803. This author's research was supported by the Louisiana Board of Regents Research Competitiveness Subprogram.

‡Institut für Informatik, Albert-Ludwigs-Universität, Georges-Köhler-Allee, 79110 Freiburg, Germany (vanstee@informatik.uni-freiburg.de). This work was done while the author was at the CWI, The Netherlands. This author's research was supported by the Netherlands Organization for Scientific Research (NWO), project SION 612-30-002.

§School of Computer Science, The Interdisciplinary Center, Herzliya, Israel (lea@idc.ac.il). This author's research was supported by Israel Science Foundation grant 250/01.

algorithms. Basically, we want to find an algorithm which incurs cost which is within a constant factor of the minimum possible cost, no matter what the input is. This constant factor is known as the asymptotic performance ratio.

A bin packing algorithm uses *bounded space* if it has only a constant number of bins available to accept items at any point during processing. These bins are called *open* bins. Bins which have already accepted some items, but which the algorithm no longer considers for packing, are *closed* bins. While bounded space algorithms are sometimes desirable, it is often the case that unbounded space algorithms can achieve lower performance ratios.

We define the asymptotic performance ratio more precisely. For a given input sequence σ, let $\text{cost}_{\mathcal{A}}(\sigma)$ be the sum of the capacities of the bins used by algorithm $\mathcal{A}$ on σ. Let $\text{cost}(\sigma)$ be the minimum possible cost to pack pieces in σ. The *asymptotic performance ratio* for an algorithm $\mathcal{A}$ is defined to be

$$R_{\mathcal{A}}^{\infty} = \limsup_{n\to\infty} \max_{\sigma} \left\{ \frac{\text{cost}_{\mathcal{A}}(\sigma)}{\text{cost}(\sigma)} \;\middle|\; \text{cost}(\sigma) = n \right\}.$$

The *optimal asymptotic performance ratio* is defined to be

$$R_{\text{OPT}}^{\infty} = \inf_{\mathcal{A}} R_{\mathcal{A}}^{\infty}.$$

Our goal is to find an algorithm with asymptotic performance ratio close to R_{OPT}^{∞}.

Previous results. The online bin packing problem was first investigated by Johnson [9, 10]. He showed that the NEXT FIT algorithm has performance ratio 2. Subsequently, it was shown by Johnson et al. that the FIRST FIT algorithm has performance ratio $\frac{17}{10}$ [11]. Yao showed that REVISED FIRST FIT has performance ratio $\frac{5}{3}$ and further showed that no online algorithm has performance ratio less than $\frac{3}{2}$ [21]. Brown [1] and Liang [14] independently improved this lower bound to 1.53635. This was subsequently improved by van Vliet to 1.54014 [19]. Chandra [2] shows that the preceding lower bounds also apply to randomized algorithms.

Define

$$u_{i+1} = u_i(u_i - 1) + 1, \qquad u_1 = 2,$$

and

$$h_{\infty} = \sum_{i=1}^{\infty} \frac{1}{u_i - 1} \approx 1.69103.$$

Lee and Lee showed that the HARMONIC algorithm, which uses bounded space, achieves a performance ratio arbitrarily close to h_{∞} [13]. They further showed that no bounded space online algorithm achieves a performance ratio less than h_{∞} [13]. A sequence of further results has brought the upper bound down to 1.58889 [13, 15, 16, 17].

The variable-sized bin packing problem was first investigated by Friesen and Langston [7, 8]. Kinnersley and Langston gave an online algorithm with performance ratio $\frac{7}{4}$ [12]. Csirik proposed the VARIABLE HARMONIC algorithm and showed that it has performance ratio at most h_{∞} [4]. This algorithm is based on the HARMONIC algorithm of Lee and Lee [13]. Like HARMONIC, it uses bounded space. Csirik also showed that if the algorithm has two bin sizes 1 and $\alpha < 1$ and if it is allowed to pick α, then a performance ratio of $\frac{7}{5}$ is possible [4]. Seiden has recently shown that VARIABLE HARMONIC is an optimal bounded-space algorithm [18].

The related problem of variable-sized bin covering has been solved by Woeginger and Zhang [20] and extended by Epstein [6].

Our results. In this paper, we present new algorithms for the variable-sized online bin packing problem. By combining the upper bounds for these algorithms, we improve the upper bound for this problem from 1.69103 to 1.63597. Our technique extends the general packing algorithm analysis technique developed by Seiden [17]. We also show the first lower bounds for variable-sized online bin packing. We focus on the case in which there are two bin sizes. However, our techniques are applicable to the general case. We think that our results are particularly interesting because of the unusual fractal-like curves that arise in the investigation of our algorithms and lower bounds.

2. Upper bounds. To begin, we present two different online algorithms for variable-sized bin packing.

We focus in on the case in which there are two bin sizes, $\alpha_1 < 1$ and $\alpha_2 = 1$, and examine how the performance ratios of our algorithms change as a function of α_1. Since it is understood that $m = 2$, we abbreviate α_1 using α. Both of our algorithms are combinations of the HARMONIC and REFINED HARMONIC algorithms. Both have a real parameter $\mu \in (\frac{1}{3}, \frac{1}{2})$. We call these algorithms VRH1(μ) and VRH2(μ). VRH1(μ) is defined for all $\alpha \in (0, 1)$, but VRH2(μ) is defined only for

$$\alpha > \max\left\{\frac{1}{2(1-\mu)}, \frac{1}{3\mu}\right\}. \tag{2.1}$$

First, we describe VRH1(μ). Define $n_1 = 50$, $n_2 = \lfloor n_1\alpha \rfloor$, $\epsilon = 1/n_1$, and

$$T = \left\{\frac{1}{i} \;\middle|\; 1 \le i \le n_1\right\} \cup \left\{\frac{\alpha}{i} \;\middle|\; 1 \le i \le n_2\right\} \cup \{\mu, 1-\mu\}.$$

Define $n = |T|$. Note that it may be that $n < n_1 + n_2 + 2$ since T is not a multiset. Rename the members of T as $t_1 = 1 > t_2 > t_3 > \cdots > t_n = \epsilon$. For convenience, define $t_{n+1} = 0$. The interval I_j is defined to be $(t_{j+1}, t_j]$ for $j = 1, \ldots, n+1$. Note that these intervals are disjoint and that they cover $(0, 1]$. A piece of size s has *type* j if $s \in I_j$. Define the *class* of an interval I_j to be α if $t_j = \alpha/k$ for some positive integer k; otherwise, the class is 1.

The basic idea of VRH1 is as follows: When each piece arrives, we determine the interval I_j to which it belongs. If this is a class 1 interval, we pack the item in a size 1 bin using a variant of REFINED HARMONIC. If it is a class α interval, we pack the item in a size α bin using a variant of HARMONIC.

VRH1 packs bins in *groups*. All the bins in a group are packed in a similar fashion. The groups are determined by the set T. We define

$$g = \begin{cases} 3 & \text{if } \alpha > 1-\mu, \\ 2 & \text{otherwise.} \end{cases} \qquad h = \begin{cases} 6 & \text{if } \alpha/2 > \mu, \\ 5 & \text{if } \alpha > \mu \text{ and } \alpha/2 \le \mu, \\ 4 & \text{otherwise.} \end{cases}$$

Note that these functions are defined so that $t_g = 1 - \mu$ and $t_h = \mu$. The groups are named $(g, h), 1, \ldots, g-1, g+1, g+2, \ldots, n$.

Bins in group $j \in \{1, 2, \ldots, n\} \setminus \{g\}$ contain only type j pieces.

Bins in group (g, h) all have capacity 1. Closed bins contain one type g piece and one type h piece.

```
VRH1
        Initialize x ← 0 and y ← 0.
        For each item p:
            j ← type of p.
            If j = n then pack p using NEXT FIT in a group n bin.
            Else, if j = g then PUT(p, (g, h)).
            Else, if j = h:
                x ← x + 1.
                If y < ⌊τx⌋:
                    y ← y + 1.
                    PUT(p, (g, h)).
                Else PUT(p, h).
            Else PUT(p, j).
```

```
PUT(p, G)
        If there is no open bin in G then allocate a new bin b.
        Else, let b be an arbitrary open bin in G.
        Pack p in b.
```

FIG. 2.1. *The* VRH1(μ) *algorithm and the* PUT *subroutine.*

Bins in group n all have capacity 1 and are packed using the NEXT FIT algorithm. There is always one open bin in group n. When a type n piece arrives, if the piece fits in the open bin, it is placed there. If not, the open bin is closed, the piece is placed in a newly allocated open group n bin.

For group $j \in \{1, 2, \ldots, n-1\} \setminus \{g\}$, the capacity of bins in the group depends on the class of I_j. If I_j has class 1, then each bin has capacity 1, and each closed bin contains $\lfloor 1/t_j \rfloor$ items of type j. Note that t_j is the reciprocal of an integer for $j \neq h$, and therefore $\lfloor 1/t_j \rfloor = 1/t_j$. If I_j has class α, then each bin has capacity α, and each closed bin contains $\lfloor \alpha/t_j \rfloor$ items of type j. Similarly to before, t_j/α is the reciprocal of an integer, and therefore $\lfloor \alpha/t_j \rfloor = \alpha/t_j$. For each of these groups, there is at most one open bin.

The algorithm has a real parameter $\tau \in [0, 1]$, which for now we fix to be $\frac{1}{7}$. Essentially, a proportion τ of the type h items are reserved for placement with type g items.

A precise definition of VRH1 appears in Figure 2.1. The algorithm uses the subroutine PUT(p, G), where p is an item and G is a group.

We analyze VRH1 using the technique of *weighting systems* introduced in [17]. A weighting system is a tuple $(\mathbb{R}^\ell, \mathbf{w}, \xi)$, where $\mathbb{R}^\ell$ is a real vector space, $\mathbf{w}$ is a *weighting function*, and ξ is a *consolidation function*. We shall simply describe the weighting system for VRH1 and assure the reader that our definitions meet the requirements put forth in [17].

For VRH1, we use $\ell = 3$ and define $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ to be orthogonal unit basis vectors.

The weighting function is

$$\mathbf{w}(x) = \begin{cases} \mathbf{b} & \text{if } x \in I_g, \\ (1-\tau)\dfrac{\mathbf{a}}{2} + \tau\,\mathbf{c} & \text{if } x \in I_h, \\ \dfrac{\mathbf{a}\,x}{1-\epsilon} & \text{if } x \in I_n, \\ \mathbf{a}\,t_i & \text{otherwise.} \end{cases}$$

The consolidation function is $\xi(x\,\mathbf{a} + y\,\mathbf{b} + z\,\mathbf{c}) = x + \max\{y, z\}$. The following lemma allows us to upper bound the performance of VRH1 using the preceding weighting system.

LEMMA 2.1. *For all input sequences σ,*

$$\text{cost}_{\text{VRH1}}(\sigma) \le \xi\left(\sum_{i=1}^{n} \mathbf{w}(p_i)\right) + O(1).$$

Proof. We count the cost for bins in each group.

First, consider bins in group n. Each of these is packed using NEXT FIT and contains only pieces of size at most ϵ. By the definition of NEXT FIT, each closed bin contains items of total size at least $1-\epsilon$, and there is at most one open bin. Therefore, the number of bins used is at most

$$\frac{1}{1-\epsilon}\sum_{p_i \in I_n} p_i + 1 = \mathbf{a} \cdot \sum_{p_i \in I_n} \mathbf{w}(p_i) + O(1).$$

Now consider group j with $j \notin \{h, (g,h), n\}$. There is at most one open bin in this group. The capacity x of each bin is equal to the class of I_j. The number of items in each closed bin is $\lfloor x/t_j \rfloor$. Since $j \notin \{h, (g,h), n\}$, we have $\lfloor x/t_j \rfloor = x/t_j$. Putting these facts together, the cost is at most

$$\sum_{p_i \in I_j} \frac{x}{\lfloor x/t_j \rfloor} + 1 = \sum_{p_i \in I_j} t_j + 1 = \mathbf{a} \cdot \sum_{p_i \in I_j} \mathbf{w}(p_i) + O(1).$$

Next, consider group h. Let k be the number of type h items in σ. The algorithm clearly maintains the invariant that $\lfloor \tau k \rfloor$ of these items go to group (g,h). The remainder are packed two to a bin in capacity 1 bins. At most one bin in group h is open. The total is at most

$$\frac{k - \lfloor \tau k \rfloor}{2} + 1 = \sum_{p_i \in I_h} \frac{1-\tau}{2} + O(1) = \mathbf{a} \cdot \sum_{p_i \in I_h} \mathbf{w}(p_i) + O(1).$$

Finally, consider group (g,h). Let f be the number of type g items in σ. The number of bins is

$$\max\{f, \lfloor \tau k \rfloor\} = \max\{f, \tau k\} + O(1) = \max\left\{\mathbf{b} \cdot \sum_{p_i \in I_g} \mathbf{w}(p_i),\ \mathbf{c} \cdot \sum_{p_i \in I_h} \mathbf{w}(p_i)\right\} + O(1).$$

Putting all these results together, the total cost is at most

$$\mathbf{a} \cdot \sum_{i=1}^{n} \mathbf{w}(p_i) + \max\left\{\mathbf{b} \cdot \sum_{i=1}^{n} \mathbf{w}(p_i),\ \mathbf{c} \cdot \sum_{i=1}^{n} \mathbf{w}(p_i)\right\} + O(1) = \xi\left(\sum_{i=1}^{n} \mathbf{w}(p_i)\right) + O(1). \qquad \square$$

From [17], we also have the following lemma.

LEMMA 2.2. *For any input σ on which* VRH1 *achieves a performance ratio of c, there exists an input σ' where* VRH1 *achieves a performance ratio of at least c and*

1. *every bin in an optimal solution is full, and*
2. *every bin in some optimal solution is packed identically.*

Given these two lemmas, the problem of upper bounding the performance ratio of VRH1 is reduced to that of finding the single packing of an optimal bin with maximal weight/size ratio. We consider the following integer program: Maximize $\xi(\mathbf{x})/\beta$ subject to

$$\mathbf{x} = \mathbf{w}(y) + \sum_{j=1}^{n-1} q_j \mathbf{w}(t_j); \tag{2.2}$$

$$y = \beta - \sum_{j=1}^{n-1} q_j \, t_{j+1}, \tag{2.3}$$

$$y > 0, \tag{2.4}$$

$$q_j \in \mathbb{N} \qquad \text{for } 1 \le j \le n-1, \tag{2.5}$$

$$\beta \in \{1, \alpha\}, \tag{2.6}$$

over variables $\mathbf{x}, y, \beta, q_1, \ldots, q_{n-1}$. Intuitively, q_j is the number of type j pieces in an optimal bin. y is an upper bound on space available for type n pieces. Note that strict inequality is required in (2.4) because a type j piece is strictly larger than t_{j+1}. Call this integer linear program $\mathcal{P}$. The value of $\mathcal{P}$ upper bounds the asymptotic performance ratio of VRH1.

The value of $\mathcal{P}$ is easily determined using a branch and bound procedure very similar to those in [17, 18]. Define

$$\psi_i = \max\left\{(\mathbf{a}+\mathbf{b}+\mathbf{c})\cdot\mathbf{w}(t_i),\ \frac{1}{1-\epsilon}\right\} \qquad \text{for } 1 \le i \le n-1; \qquad \psi_n = \frac{1}{1-\epsilon}.$$

Intuitively, ψ_i is the maximum contribution to the objective function for a type i item relative to its size. We define π so that

$$\psi_{\pi(1)} \ge \psi_{\pi(2)} \ge \cdots \ge \psi_{\pi(n)}.$$

The procedure is displayed in Figure 2.2. The heart of the procedure is the subroutine TRYALL, which basically finds the maximum weight which can be packed into a bin of size β. Using π, we try first to include items which contribute the most to the objective relative to their size. This is a heuristic. The variables $\mathbf{v}$ and y keep track of the weight and total size of items included so far. The variable j indicates that the current item type is $\pi(j)$. In the For loop at the end of TRYALL, we try each possible number of type $\pi(j)$ items, starting with the largest possible number. First packing as many items as possible is a heuristic which seems to speed up computation. The current maximum is stored in x. When we enter TRYALL, we first compute an upper bound given the packing so far, which is stored in z. When $j = n$, this upper bound is exactly the objective value. If $z \le x$, we do not have to consider any packing reachable from the current one, and we drop straight through. In the main routine, we simply initialize x, call TRYALL for the two bin sizes, and return x.

Now we describe VRH2(μ). Redefine

$$T = \left\{\frac{1}{i} \;\middle|\; 1 \le i \le n_1\right\} \cup \left\{\frac{\alpha}{i} \;\middle|\; 1 \le i \le n_2\right\} \cup \{\alpha\mu,\ \alpha(1-\mu)\}.$$

```
x ← 1.
TRYALL(1, 0, 1, 1).
TRYALL(1, 0, α, α).
Return x.
```

```
TRYALL(j, v, y, β)
    z ← (ξ(v) + y ψ_{π(j)}) /β.
    If z > x then:
    If j = n then:
        x ← z.
    Else:
        For i ← ⌈y/t_{π(j)+1}⌉ − 1, ..., 0:
            TRYALL(j + 1, v + i w(t_{π(j)}), y − i t_{π(j)+1}, β).
```

FIG. 2.2. *The algorithm for computing $\mathcal{P}$ along with subroutine* TRYALL.

Define n_1, n_2, ϵ, and n as for VRH1. Again, rename the members of T as $t_1 = 1 > t_2 > t_3 > \cdots > t_n = \epsilon$. Equation (2.1) guarantees that $1/2 < \alpha(1-\mu) < \alpha < 1$ and $1/3 < \alpha\mu < \alpha/2 < 1/2$, so we have $g = 3$ and $h = 6$. The only difference from VRH1 is that (g, h) bins have capacity α. Otherwise, the two algorithms are identical. We therefore omit a detailed description and analysis of VRH2.

We display the upper bound on the performance ratio achieved using the best of VRH1(μ), VRH2(μ), and VARIABLE HARMONIC in Figure 4.3. This upper bound is achieved by optimizing μ for each choice of α. Our upper bound is at most $\frac{373}{228} < 1.63597$ for all α, which is the performance ratio of REFINED HARMONIC in the classic bin packing context.

3. Lower bounds. We now consider the question of lower bounds. Prior to this work, no general lower bounds for variable-sized online bin packing were known.

Our method follows that of Brown [1], Liang [14], and van Vliet [19]. We give some unknown online bin packing algorithm $\mathcal{A}$ one of k possible different inputs. These inputs are defined as follows: Let $\varrho = s_1, s_2, \ldots, s_k$ be a sequence of *item sizes* such that $0 < s_1 < s_2 < \cdots < s_k \leq 1$. Let ϵ be a small positive constant. We define σ_0 to be the empty input. Input σ_i consists of σ_{i-1} followed by n items of size $s_i + \epsilon$. Algorithm $\mathcal{A}$ is given σ_i for some $i \in \{1, \ldots, k\}$.

A *pattern* with respect to ϱ is a tuple $p = \langle \text{size}(p), p_1, \ldots, p_k \rangle$, where size($p$) is a positive real number and p_i, $1 \leq i \leq k$, are nonnegative integers such that

$$\sum_{i=1}^{k} p_i \, s_i < \text{size}(p).$$

Intuitively, a pattern describes the contents of some bin of capacity size(p). Define $\mathcal{P}(\varrho, \beta)$ to be the set of all patterns p with respect to ϱ with size(p) = β. Further define

$$\mathcal{P}(\varrho) = \bigcup_{i=1}^{m} \mathcal{P}(\varrho, \alpha_i).$$

Note that $\mathcal{P}(\varrho)$ is necessarily finite. Given an input sequence of items, an algorithm is defined by the numbers and types of items it places in each of the bins it uses.

Specifically, any algorithm is defined by a function $\Phi : \mathcal{P}(\varrho) \mapsto \mathbb{R}_{\geq 0}$. The algorithm uses $\Phi(p)$ bins containing items as described by the pattern p. We define $\phi(p) = \Phi(p)/n$.

Consider the function Φ that determines the packing used by online algorithm $\mathcal{A}$ for σ_k. Since $\mathcal{A}$ is online, the packings it uses for $\sigma_1, \ldots, \sigma_{k-1}$ are completely determined by Φ. We assign to each pattern a *class*, which is defined as

$$\text{class}(p) = \min\{i \mid p_i \neq 0\}.$$

Intuitively, the class tells us the first sequence σ_i, which results in some item being placed into a bin packed according to this pattern. That is, if the algorithm packs some bins according to a pattern which has class i, then these bins will contain one or more items after σ_i. Define

$$\mathcal{P}_i(\varrho) = \{p \in \mathcal{P}(\varrho) \mid \text{class}(p) \leq i\}.$$

Then, if $\mathcal{A}$ is determined by Φ, its cost for σ_i is simply

$$n \sum_{p \in \mathcal{P}_i(\varrho)} \text{size}(p)\phi(p).$$

Since the algorithm must pack every item, we have the following constraints:

$$n \sum_{p \in \mathcal{P}(\varrho)} \phi(p)\, p_i \geq n \qquad \text{for } 1 \leq i \leq k.$$

For a fixed n, define $\chi_i(n)$ to be the optimal offline cost for packing the items in σ_i. The following lemma gives us a method of computing the optimal offline cost for each sequence.

LEMMA 3.1. *For $1 \leq i \leq k$, $\chi^* = \lim_{n\to\infty} \chi_i(n)/n$ exists and is the value of the following linear program: Minimize*

$$\sum_{p \in \mathcal{P}_i(\varrho)} \text{size}(p)\phi(p) \tag{3.1}$$

subject to

$$1 \quad \leq \quad \sum_{p \in \mathcal{P}(\varrho)} \phi(p)\, p_j \qquad \text{for } 1 \leq j \leq i \tag{3.2}$$

over variables χ_i and $\phi(p)$, $p \in \mathcal{P}(\varrho)$.

Proof. Clearly, the linear program always has a finite value between $\sum_{j=1}^{i} s_j$ and i. For any fixed n, the optimal offline solution is determined by some ϕ. It must satisfy the constraints of the linear program, and the objective value is exactly the cost incurred. Therefore, the linear program lower bounds the optimal offline cost. The linear program is a relaxation in that it allows a fractional number of bins of any pattern, whereas a legitimate solution must have an integral number. Rounding the relaxed solution up to get a legitimate one, the change in the objective value is at most $|\mathcal{P}(\varrho)|/n$. □

Given the construction of a sequence, we need to evaluate

$$c = \min_{\mathcal{A}} \max_{i=1,\ldots,k} \limsup_{n\to\infty} \frac{\text{cost}_{\mathcal{A}}(\sigma_i)}{\chi_i(n)}.$$

As $n \to \infty$, we can replace $\chi_i(n)/n$ by χ_i^*. Once we have the values $\chi_1^*, \ldots, \chi_k^*$, we can readily compute a lower bound for our online algorithm.

LEMMA 3.2. *The optimal value of the linear program: Minimize c subject to*

$$\begin{aligned} c &\geq \frac{1}{\chi_i^*} \sum_{p \in \mathcal{P}_i(\varrho)} \text{size}(p)\phi(p) \qquad \text{for } 1 \leq i \leq k, \\ 1 &\leq \sum_{p \in \mathcal{P}(\varrho)} \phi(p)\, p_i \qquad \text{for } 1 \leq i \leq k \end{aligned} \tag{3.3}$$

over variables c and $\phi(p)$, $p \in \mathcal{P}(\varrho)$, is a lower bound on the asymptotic performance ratio of any online bin packing algorithm.

Proof. For any fixed n, any algorithm $\mathcal{A}$ has some Φ which must satisfy the second constraint. Further, Φ should assign an integral number of bins to each pattern. However, this integrality constraint is relaxed, and $\sum_{p \in \mathcal{P}_i(\varrho)} \text{size}(p)\phi(p)$ is $1/n$ times the cost to $\mathcal{A}$ for σ_i as $n \to \infty$. The value of c is just the maximum of the performance ratios achieved on $\sigma_1, \ldots, \sigma_k$. ∎

Although this is essentially the result we seek, a number of issues are left to be resolved.

The first is that these linear programs have a variable for each possible pattern. The number of such patterns is potentially quite large, and we would like to reduce the linear program size if possible. We show that this goal is indeed achievable. We say that a pattern p of class i is *dominant* if

$$s_i + \sum_{j=1}^{k} p_j\, s_j > \text{size}(p).$$

Let p be a nondominant pattern with class i. There exists a unique dominant pattern q of class i such that $p_j = q_j$ for all $i \neq j$. We call q the *dominator* of p with respect to class i.

LEMMA 3.3. *In computing the values of the linear programs in Lemmas* 3.1 *and* 3.2*, it suffices to consider only dominant patterns.*

Proof. We transform a linear program solution by applying the following operation to each nondominant pattern p of class i: Let $x = \phi(p)$ in the original solution. We set $\phi(p) = 0$ and increment $\phi(q)$ by x, where q is the dominator of p with respect to i. The new solution remains feasible, and its objective value has not changed. Further, the value of $\phi(p)$ is zero for every nondominant p; therefore, these variables can be safely deleted. ∎

Given a sequence of item sizes ϱ, we can compute a lower bound $L_m(\varrho, \alpha_1, \ldots, \alpha_{m-1})$ using the following algorithm:

1. Enumerate the dominant patterns.
2. For $1 \leq i \leq k$, compute χ_i via the linear program given in Lemma 3.1.
3. Compute and return the value of the linear program given in Lemma 3.2.

Step 1 is most easily accomplished via a simple recursive function. Our concern in the remainder of the paper shall be to study the behavior of $L_m(\varrho, \alpha_1, \ldots, \alpha_{m-1})$ as a function of ϱ and $\alpha_1, \ldots, \alpha_{m-1}$.

4. Lower bound sequences. Up to this point, we have assumed that we were given some fixed item sequence ϱ. We consider now the question of choosing ϱ. We again focus on the case in which there are two bin sizes and examine properties of $L_2(\varrho, \alpha_1)$. We again abbreviate α_1 using α and L_2 using L.

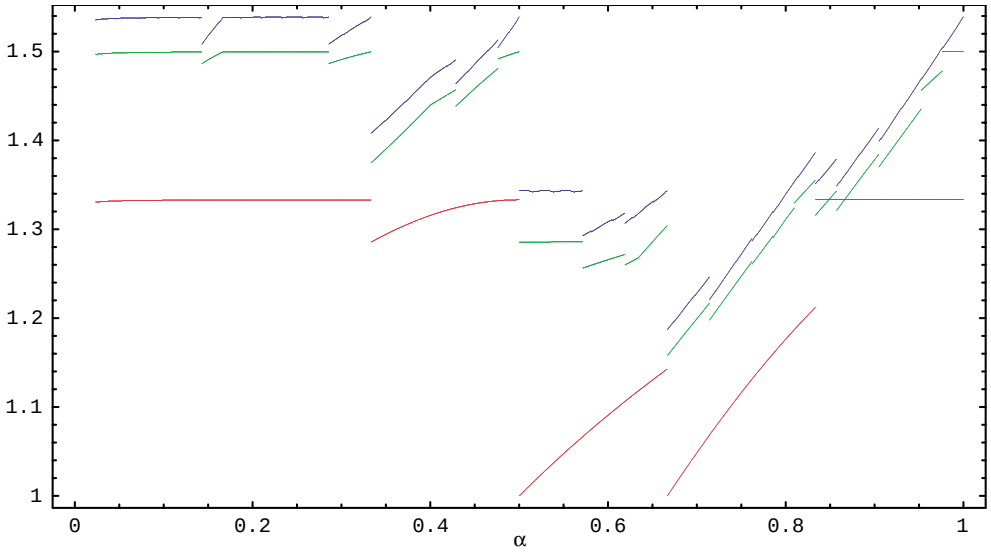


FIG. 4.1. *The evolution of the curves given by the greedy item sequence. The lowest curve is* $\frac{1}{2}, \frac{1}{3}$; *the middle curve is* $\frac{1}{2}, \frac{1}{3}, \frac{1}{7}$; *the highest curve is* $\frac{1}{2}, \frac{1}{3}, \frac{1}{7}, \frac{1}{43}$.

To begin, we define the idea of a *greedy* sequence. Let ϵ denote the empty sequence, and let $\wedge$ denote the sequence concatenation operator. The greedy sequence $\Gamma_\tau(\beta)$ for capacity β with cutoff τ is defined by

$$\gamma(\beta) = \frac{1}{\lfloor \frac{1}{\beta} \rfloor + 1}, \qquad \Gamma_\tau(\beta) = \begin{cases} \epsilon & \text{if } \beta < \tau, \\ \gamma(\beta) \wedge \Gamma_\tau(\beta - \gamma(\beta)) & \text{otherwise.} \end{cases}$$

The sequence defines the item sizes which would be used if we packed a bin of capacity β using the following procedure: At each step, we determine the remaining capacity in our bin. We choose as the next item the largest reciprocal of an integer which fits without using the remaining capacity completely. We stop when the remaining capacity is smaller than τ. Note that, for $\tau = 0$, we get the infinite sequence. We shall use Γ as a shorthand for Γ_0.

The recurrence u_i described in section 1, which is found in connection with bounded-space bin packing [13], gives rise to the sequence

$$\frac{1}{u_i} = \frac{1}{2}, \frac{1}{3}, \frac{1}{7}, \frac{1}{43}, \frac{1}{1807}, \ldots.$$

This turns out to be the infinite greedy sequence $\Gamma(1)$. Somewhat surprisingly, it is also the sequence used by Brown [1], Liang [14], and van Vliet [19] in the construction of their lower bounds. In essence, they analytically determine the value of $L_1(\Gamma_\tau(1))$. Liang and Brown lower bound the value, while van Vliet determines it exactly.

This well-known sequence is our first candidate. Actually, we use the first k item sizes in it, and we resort them so that the algorithm is confronted with items from smallest to largest. In general, this resorting seems to be a good heuristic since the algorithm has the most decisions to make about how the smallest items are packed but, on the other hand, has the least information about which further items will be received. The results are shown in Figure 4.1.

Examining Figure 4.1, one immediately notices that $L(\Gamma_\tau(1), \alpha)$ exhibits some very strange behavior. The curve is highly discontinuous. Suppose we have a finite sequence ϱ, where each item size is a continuous function of $\alpha \in (0,1)$. Tuple p is a *potential pattern* if there exists an $\alpha \in (0,1)$ such that p is a pattern. The set of breakpoints of p with respect to ϱ is defined to be

$$B(p,\varrho) = \left\{ \alpha \in (0,1) \;\middle|\; \sum_{i=1}^{k} p_i\, s_i = \text{size}(p) \right\}.$$

Let $\mathcal{P}^*$ be the set of all potential patterns. The set of all breakpoints is

$$B(\varrho) = \bigcup_{p \in \mathcal{P}^*} B(p,\varrho).$$

Intuitively, at each breakpoint, some combinatorial change occurs, and the curve may jump. In the intervals between breakpoints, the curve behaves nicely as summarized by the following lemma.

LEMMA 4.1. *Let ϱ be a finite item sequence with each item size a continuous function of $\alpha \in (0,1)$. In any interval $I = (\ell, h)$ which does not contain a breakpoint, $L(\varrho,\alpha)$ is continuous. Furthermore, for all $\alpha \in I$,*

$$L(\varrho,\alpha) \geq \min\left\{ \frac{\ell+h}{2h}, \frac{2\ell}{\ell+h} \right\} L\left(\varrho, \tfrac{1}{2}(\ell+h)\right).$$

This lemma follows as a corollary from the following lemma.

LEMMA 4.2. *Let ϱ be a finite item sequence with each item size a continuous function of $\alpha \in (0,1)$. Let I be any interval which does not contain a breakpoint, and let α be any point in I. The following two results hold:*

1. *If $\delta > 0$ is such that $\alpha + \delta \in I$, then*

$$L(\varrho,\alpha+\delta) \geq \left(1 - \frac{\delta}{\alpha+\delta}\right) L(\varrho,\alpha).$$

2. *If $\delta > 0$ is such that $\alpha - \delta \in I$, then*

$$L(\varrho,\alpha-\delta) \geq \left(1 - \frac{\delta}{\alpha}\right) L(\varrho,\alpha).$$

Proof. We first prove statement 1. Denote by $\chi_i^*(x)$ the value of χ_i^* at $\alpha = x$. For $1 \leq i \leq k$, we have

$$\chi_i^*(\alpha+\delta) \leq \frac{\alpha+\delta}{\alpha} \chi_i^*(\alpha).$$

To see this, note that any feasible Φ at α is also feasible at $\alpha+\delta$ since both points are within I and (3.2) does not change within this interval. Each term in (3.1) increases by at most $(\alpha+\delta)/\alpha$. Now consider the linear program of Lemma 3.2. Consider some arbitrary feasible solution ϕ at α. At $\alpha+\delta$, this solution is still feasible (except that possibly c must increase). In the sum $1/\chi_i^* \sum_{p \in \mathcal{P}_i(\varrho)} \text{size}(p)\phi(p)$, the factor $1/\chi_i^*$ decreases by at most $\alpha/(\alpha+\delta)$, and size(p) cannot decrease.

Now consider statement 2. The arguments are quite similar. For $1 \leq i \leq k$, we have

$$\chi_i^*(\alpha-\delta) \leq \chi_i^*(\alpha).$$

Again, a feasible solution remains feasible. Further, its objective value (3.1) cannot increase. Considering the linear program of Lemma 3.2, we find that, for each feasible solution, each sum $1/\chi_i^* \sum_{p\in\mathcal{P}_i(\varrho)} \text{size}(p)\phi(p)$ decreases by a factor of at most $(\alpha-\delta)/\alpha$. □

Considering Figure 4.1 again, there are sharp drops in the lower bound near the points $\frac{1}{3}$, $\frac{1}{2}$, and $\frac{2}{3}$. It is not hard to see why the bound drops so sharply at those points. For instance, if α is just larger than $\frac{1}{2}+\epsilon$, then the largest items in $\Gamma(1)$ can each be put in their own bin of size α. If $\alpha \geq \frac{2}{3}+2\epsilon$, two items of size $\frac{1}{3}+\epsilon$ can be put pairwise in bins of size α. In short, in such cases, the online algorithm can pack some of the largest elements in the list with very little wasted space—hence the low resulting bound.

This observation leads us to try other sequences in which the last items cannot be packed well. A first candidate is the sequence $\alpha, \Gamma(1-\alpha)$. As expected, this sequence performs much better than $\Gamma(1)$ in the areas described above.

It is possible to find further improvements for certain values of α. For instance, the sequence $\alpha/2, \Gamma(1-\alpha/2)$ also works well in some places, and we used other sequences as well. We give two examples in Figure 4.2.

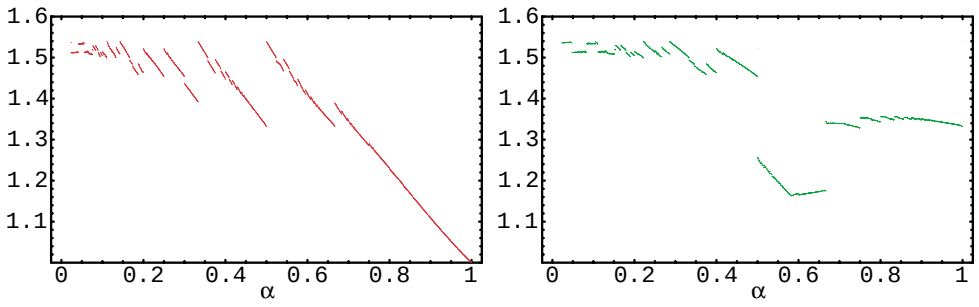


FIG. 4.2. *Two lower bound sequences for* $\tau = 1/1000$: *On the left is* $\alpha, \Gamma_\tau(1-\alpha)$, *and on the right is* $\frac{\alpha}{2}, \Gamma_\tau(1-\frac{\alpha}{2})$.

As a general guideline for finding sequences, items should not fit too well in either bin size. If an item has size x, then $\min\{1 - \lfloor \frac{1}{x} \rfloor x,\ \alpha - \lfloor \frac{\alpha}{x} \rfloor x\}$ should be as large as possible. In areas where a certain item in a sequence fits very well, that item should be adjusted (e.g., use an item $1/(j+1)$ instead of the item $1/j$), or a completely different sequence should be used. (This helps explain why the algorithms have a low competitive ratio for α close to 0.7: in that area, this minimum is never very large.)

Furthermore, as in the classical bin packing problem, sequences that are bad for the online algorithm should have very different optimal solutions for each prefix sequence. Finally, the item sizes should not increase too quickly or too slowly: If items are very small, the smallest items do not affect the online performance much, while if items are close in size, the sequence is easy because the optimal solutions for the prefixes are alike.

In addition to the three sequences already described, namely, the greedy sequence, $\alpha, \Gamma(1-\alpha)$, and $\alpha/2, \Gamma(1-\alpha/2)$, we have found that the following sequences yield good results in restricted areas: $\alpha, \frac{1}{3}, \frac{1}{7}, \frac{1}{43}$; $\frac{1}{2}, \frac{1}{4}, \frac{1}{5}, \frac{1}{21}$; and $\frac{1}{2}, \frac{\alpha}{2}, \frac{1}{9}, \Gamma_\tau(\frac{7}{18} - \frac{\alpha}{2})$.

Using Lemma 4.2, we obtain the following main theorem of this section.

THEOREM 4.3. *Any online algorithm for the variable-sized bin packing problem with* $m = 2$ *has asymptotic performance ratio at least* $495176908800/370749511199 > 1.33561$.

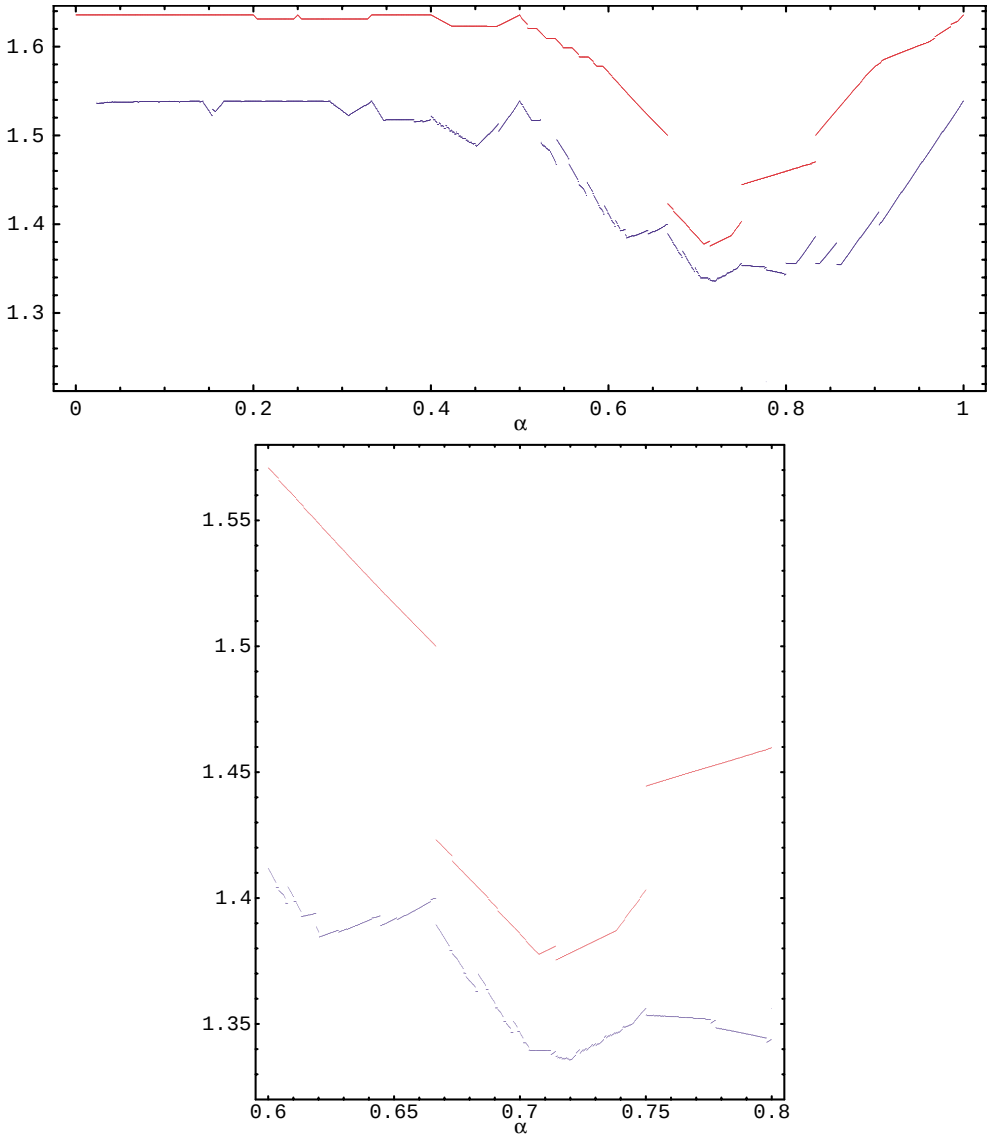


FIG. 4.3. *The best upper and lower bounds for variable-sized online bin packing. The bottom figure is a closeup of* $[.6, .8]$. *The upper bound is best of the* VRH1, VRH2, *and* VARIABLE HARMONIC *algorithms.*

Proof. First, note that, for $\alpha \in (0, 1/43]$, the sequence $\frac{1}{2}, \frac{1}{3}, \frac{1}{7}, \frac{1}{43}$ yields a lower bound of $217/141 > 1.53900$ as in the classic problem: Bins of size α are of no use.

We use the sequences described in the preceding paragraphs. For each sequence ϱ, we compute a lower bound on $(1/43, 1)$ using the following procedure.

Define $\varepsilon = 1/10000$. We break the interval $(0, 1)$ into subintervals using the lattice points $\varepsilon, 2\varepsilon, \ldots, 1 - \varepsilon$. To simplify the determination of breakpoints, we use a constant sequence for each subinterval. This constant sequence is fixed at the upper limit of the interval. That is, throughout the interval $[\ell\varepsilon, \ell\varepsilon + \varepsilon)$, we use the sequence $\varrho|_{\alpha=\ell\varepsilon+\varepsilon}$. Since the sequence is constant, a lower bound on the performance ratio of any online bin packing algorithm with $\alpha \in [\ell\varepsilon, \ell\varepsilon + \varepsilon)$ can be determined by the following algorithm:

1. $\varrho' \leftarrow \varrho|_{\alpha=\ell\varepsilon+\varepsilon}$.
2. Initialize $B \leftarrow \{\ell\varepsilon, \ell\varepsilon + \varepsilon\}$.
3. Enumerate all the patterns for ϱ' at $\alpha = \ell\varepsilon + \varepsilon$.
4. For each pattern:
   (a) $z \leftarrow \sum_{i=1}^{k} p_i\, s_i$.
   (b) If $z \in (\ell\varepsilon, \ell\varepsilon + \varepsilon)$, then $B \leftarrow B \cup \{z\}$.
5. Sort B to get $b_1, b_2, \ldots, b_j$.
6. Calculate and return the value:

$$\min_{1 \le i < j} \min \left\{ \frac{b_i + b_{i+1}}{2b_{i+1}},\ \frac{2b_i}{b_i + b_{i+1}} \right\} L\left(\varrho',\, \tfrac{1}{2}(b_i + b_{i+1})\right).$$

We implemented this algorithm in `Mathematica` and used it to find lower bounds for each of the aforementioned sequences. The results are shown in Figures 4.2 and 4.3. The lowest lower bound is 495176908800/370749511199 in the interval $[0.7196, 0.7197)$. □

5. Conclusions. We have shown new algorithms and lower bounds for variable-sized online bin packing with two bin sizes. By combining these algorithms with Variable Harmonic, choosing for each size α of the second bin the best algorithm for that size, we find an algorithm with asymptotic performance ratio of at most $\frac{373}{228} < 1.63597$ for all α. The best previous upper bound was $h_\infty \approx 1.69103$.

The largest gap between the performance of the algorithm and the lower bound is 0.18193 achieved for $\alpha = 0.9071$. The smallest gap is 0.03371 achieved for $\alpha = 0.6667$. Note that, for $\alpha \le \frac{1}{2}$, there is not much differing from the classical problem: having the extra bin size does not help the online algorithm much. To be more precise, it helps about as much as it helps the offline algorithm.

Our work raises the following questions: Is there a value of α where it is possible to design a better algorithm and show a matching lower bound? Or, can a lower bound be shown anywhere that matches an existing algorithm? Note that at the moment there is also a small gap between the competitive ratio of the best algorithm and the lower bound in the classical bin packing problem.

Another interesting open problem is analyzing variable-sized bin packing with an arbitrary number of bin sizes.

REFERENCES

[1] D. J. Brown, *A Lower Bound for On-Line One-Dimensional Bin Packing Algorithms*, Tech. report R-864, Coordinated Science Laboratory, Urbana, IL, 1979.

[2] B. Chandra, *Does randomization help in on-line bin packing?*, Inform. Process. Lett., 43 (1992), pp. 15–19.

[3] E. G. Coffman, M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin packing: A survey*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS Publishing, Boston, 1997, Chap. 2.
[4] J. Csirik, *An on-line algorithm for variable-sized bin packing*, Acta Inform., 26 (1989), pp. 697–709.
[5] J. Csirik and G. Woeginger, *On-line packing and covering problems*, in On-Line Algorithms—the State of the Art, Lecture Notes in Comput. Sci. 1442, A. Fiat and G. Woeginger, eds., Springer-Verlag, New York, 1998, pp. 147–177.
[6] L. Epstein, *On-line variable sized covering*, Inform. and Comput., 171 (2001), pp. 294–305.
[7] D. K. Friesen and M. A. Langston, *A storage-size selection problem*, Inform. Process. Lett., 18 (1984), pp. 295–296.
[8] D. K. Friesen and M. A. Langston, *Variable sized bin packing*, SIAM J. Comput., 15 (1986), pp. 222–230.
[9] D. S. Johnson, *Near-Optimal Bin Packing Algorithms*, Ph.D. thesis, MIT, Cambridge, MA, 1973.
[10] D. S. Johnson, *Fast algorithms for bin packing*, J. Comput. System Sci., 8 (1974), pp. 272–314.
[11] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, *Worst-case performance bounds for simple one-dimensional packing algorithms*, SIAM J. Comput., 3 (1974), pp. 299–325.
[12] N. Kinnersley and M. Langston, *Online variable-sized bin packing*, Discrete Appl. Math., 22 (1989), pp. 143–148.
[13] C. Lee and D. Lee, *A simple on-line bin-packing algorithm*, J. ACM, 32 (1985), pp. 562–572.
[14] F. M. Liang, *A lower bound for online bin packing*, Inform. Process. Lett., 10 (1980), pp. 76–79.
[15] P. Ramanan, D. Brown, C. Lee, and D. Lee, *On-line bin packing in linear time*, J. Algorithms, 10 (1989), pp. 305–326.
[16] M. B. Richey, *Improved bounds for harmonic-based bin packing algorithms*, Discrete Appl. Math., 34 (1991), pp. 203–227.
[17] S. S. Seiden, *On the online bin packing problem*, in Proceedings of the 28th International Colloquium on Automata, Languages and Programming, Crete, Greece, 2001, pp. 237–249.
[18] S. S. Seiden, *An optimal online algorithm for bounded space variable-sized bin packing*, SIAM J. Discrete Math., 14 (2001), pp. 458–470.
[19] A. van Vliet, *An improved lower bound for online bin packing algorithms*, Inform. Process. Lett., 43 (1992), pp. 277–284.
[20] G. Woeginger and G. Zhang, *Optimal on-line algorithms for variable-sized bin covering*, Oper. Res. Lett., 25 (1999), pp. 47–50.
[21] A. C. C. Yao, *New algorithms for bin packing*, J. ACM, 27 (1980), pp. 207–227.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 470–487

ON LOCAL SEARCH AND PLACEMENT OF METERS IN NETWORKS*

SAMIR KHULLER†, RANDEEP BHATIA‡, AND ROBERT PLESS§

Abstract. This work is motivated by the problem of placing pressure-meters in fluid networks. The problem is formally defined in graph-theoretic terms as follows. Given a graph, find a cotree (complement of a tree) incident upon the minimum number of vertices. We show that this problem is NP-hard and MAX SNP-hard. We design an algorithm with an approximation factor of $2 + \epsilon$ for this problem for any fixed $\epsilon > 0$. This approximation bound comes from the analysis of a local search heuristic, a common practical optimization technique that does not often allow formal worst-case analysis. The algorithm is made very efficient by finding restrictive definitions of the local neighborhoods to be searched. We also exhibit a polynomial time approximation scheme for this problem when the input is restricted to planar graphs.

Key words. local search, approximation algorithms, feedback sets, pressure-meters

AMS subject classifications. 90C27, 68Q25, 68R10, 05C38, 05C85

PII. S0097539799363359

1. Introduction.

To measure flow through edges of a fluid network, one can install flow-meters on all edges. However, *one does not need to install flow-meters on all the edges in the network.* By installing flow-meters on edges of any cotree[1] [16] (see Figure 1.1), we can *infer* the flow on edges of the spanning tree due to flow conservation [21]. This requires that we install exactly $|E| - (|V| - 1)$ flow-meters, where $|E|$ is the number of edges and $|V|$ is the number of vertices, since every cotree has $|E| - (|V| - 1)$ edges in a connected network. Since "the cost of flow meters is several times the cost of pressure meters" [21], another option is to install pressure-meters at nodes of the network. A pressure-meter measures fluid pressure at a node of the network, and one can *compute* the flow on an edge by measuring the pressure reading on both of the incident nodes (see [19, 20]). One option would be to place a pressure-meter at each node of the network. However, this is not essential. Since we need only to compute the flow on edges of a cotree, we are looking for a cotree that is incident on as few vertices as possible (to minimize cost). If we place pressure-meters at the nodes incident on the edges of the cotree, we can compute the flow on all cotree edges and infer the flow on the remaining edges of the network. In Figure 1.2, we show a network. Assume that we know the inflow and outflow from the network. By measuring the pressure reading at the marked vertices, we can compute the flow on the edges that are shown. Using this information, we can infer the flow on the remaining edges of the network.

*Received by the editors October 25, 1999; accepted for publication (in revised form) October 2, 2002; published electronically February 20, 2003. A preliminary version of this paper appeared in *Proceedings of the 11th Annual ACM–SIAM Symposium on Discrete Algorithms,* 2000, pp. 319–328.
http://www.siam.org/journals/sicomp/32-2/36335.html

†Computer Science Department, University of Maryland, College Park, MD 20742 (samir@cs.umd.edu). This author's research was supported by NSF award CCR-9820965 and NSF (ITR) grant CCR-0113192.

‡Bell Labs, Lucent Technologies, Murray Hill, NJ 07974 (randeep@research.bell-labs.com). This work was done while this author was at the University of Maryland, College Park.

§Computer Science Department, Washington University, St. Louis, MO 63130-4899 (pless@cs.wustl.edu). This work was done while this author was at the University of Maryland, College Park.

[1] A cotree in a connected graph is a subset of edges that is the complement of a spanning tree.

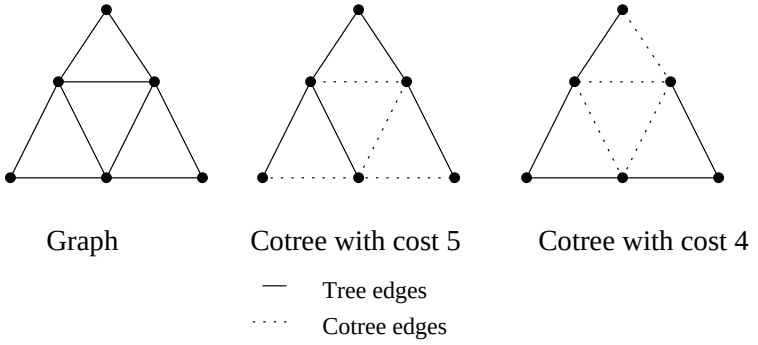


FIG. 1.1. *A simple example to show two different cotrees.*

We can also express this problem in "standard" optimization nomenclature as follows: A feedback edge set (FES) is a subset of edges in a graph whose deletion from the graph makes the graph acyclic. In other words, each cycle in the graph is required to have at least one edge from the FES. In graph-theoretic terms, minimizing the number of pressure-meters leads to the following optimization problem. The minimum vertex feedback edge set (VFES) problem is defined as follows: Given an undirected graph, find an FES incident upon the minimum number of vertices. We show that this problem is NP-hard and MAX SNP-hard and develop an approximation algorithm for it that is fast and practical and that delivers solutions that are guaranteed to have cost at most $2 + \epsilon$ times optimal (for any fixed $\epsilon > 0$).

The complement problem to the minimum VFES problem is the *full degree spanning tree* (FDST) problem [18, 21, 11, 10], namely that of computing a spanning tree T in a connected graph $G = (V, E)$ so as to *maximize* the number of vertices of *full* degree. These are vertices whose degree in the tree T is the same as their degree in the graph G. In other words, the vertices that are incident to the edges in $E \setminus T$ are exactly the vertices that do not have full degree in T. Thus, if K is the maximum number of full degree vertices in some spanning tree T in G, then $|V| - K$ is the cost of the minimum VFES. The *savings* of this solution, compared to installing pressure-meters at every vertex, is exactly the number of full degree vertices in a spanning tree since these vertices have zero degree in the corresponding FES and we do not have to install pressure-meters at these vertices. Hence maximizing the number of vertices of full degree maximizes savings. One pressure-meter is installed at each vertex that does not have full degree; by using these meters, we can compute the flow on each edge in the cotree and then infer the flow on each edge in the tree due to flow conservation.

The FDST problem was recently studied by Pothof and Schut [21], Broersma et al. [11], and Bhatia et al. [10]. The problem was introduced by Pothof and Schut [21], who gave a simple heuristic for it. In [10], it was shown that, for the case of general graphs, an approximation factor of $\Theta(\sqrt{n})$ can be obtained using a very simple algorithm called the *Greedy Star-Insertion Algorithm*. Using Håstad's result on the hardness of approximating clique, it was also shown that, if there is a polynomial time approximation algorithm for the FDST problem with a factor of $O(n^{\frac{1}{2}-\epsilon})$, then $NP = coR$ [10]. For the case of planar graphs, a polynomial time approximation scheme (PTAS) was presented [10]. Independently, Broersma et al. [11] developed a PTAS for planar graphs and showed that, for special classes of graphs such as bounded treewidth graphs and cocomparability graphs, the problem can be solved optimally

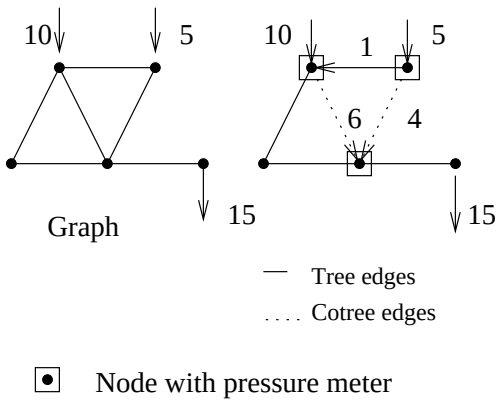


FIG. 1.2. *Figure to illustrate pressure-meters.*

in polynomial time.

Local search is widely used as a general approach to tackle hard optimization problems. Typically, each instance of the optimization problem is associated with a set of feasible solutions, together with a cost function, and the objective is to find a solution of minimum cost. To derive a local search algorithm, one superimposes a "local search neighborhood"; i.e., for each solution, we define a neighborhood of solutions that are "close" to it. We start with some initial solution. The algorithm iteratively checks to see whether or not we can "move" from the current solution to another solution in its neighborhood with lower cost, and if so, we move to the lower cost solution until we reach a locally optimal solution, a solution whose cost is as low as any other in its local search neighborhood.

Even though local search is a powerful practical tool for optimization, there are not many problems for which local search leads to good worst-case approximation factors. In fact, often, when it does lend itself to worst-case analysis, the bounds that we obtain are significantly worse than the worst-case approximation bounds that we can obtain by other methods. However, since local search is a powerful optimization method and relatively simple to implement, it is interesting to analyze local search algorithms as they enhance our understanding of the problems we are studying. For example, for the traveling salesman problem (with triangle inequality), an analysis of the local search method done by Chandra, Karloff, and Tovey [12] shows an approximation factor of $4\sqrt{n}$ for 2-exchange, with improvements for Euclidean instances.

A feedback vertex set (FVS) is a subset of vertices in the graph whose deletion from the graph makes the graph acyclic. In other words, each cycle in the graph is required to have at least one vertex from the FVS. Minimum vertex feedback set problems are NP-hard, and approximation algorithms for them have been widely studied. The first constant approximation algorithm for unweighted FVSs was given by Bar-Yehuda et al. [5]. They also gave a constant approximation algorithm for the weighted case for planar graphs and an $O(\log n)$ approximation algorithm for general graphs. For planar graphs, there is a constant factor approximation algorithm due to Goemans and Williamson [15] and recently a PTAS due to Kleinberg and Kumar [17]. For general graphs (weighted case), a 2 approximation algorithm was developed by Becker and Geiger [6] and Bafna, Berman, and Fujito [3] (see also Chudak et al. [13]).

On the other hand, for undirected graphs, the FES problem has largely been ignored since an FES is the complement of a spanning tree (cotree) and both the

maximization and minimization versions can be solved in polynomial time. However, in our problem, we need to find an FES so as to *minimize the number of vertices incident on the edges* in the FES. We show that this problem is NP-hard as well as MAX SNP-hard. Approximation algorithms for feedback arc problems on directed graphs were given by Even et al. [14].

We study the minimum VFES problem and show that the local search method applied to this problem yields a good approximation algorithm. The local search neighborhood for a given FES is defined by a constant k that allows us to "move" to any other FES that differs from the first one on fewer than k edges. The main difficulty with making this approach practical is that we need to examine all possible changes, and this is not practical even for small values of k. The naive upper bound on the size of the neighborhood we have to search for an improvement is $O(m^k n^k)$, where m is the number of edges in the graph and n is the number of vertices. However, quite surprisingly, we are able to show that we can refine the search so that we do not need to search the entire neighborhood for a better solution. We can implement each step of the local search in time $O(n^2 + nf(k))$, where $f(k)$ is a function only of k and not of the size of the graph. This makes the local search practical. Moreover, we can show that the cost of the solution produced by local search is at most $2 + \frac{1}{k}$ times the cost of the optimal solution. (In fact, we found that even for $k = 2$ this led to a very fast algorithm that we tested on several random planar and nonplanar graphs generated using the Stanford Graphbase and LEDA [9]. In these instances, the algorithm found the optimal solution in each case.)

2. Main results. We show that the minimum VFES problem is NP-hard and MAX SNP-hard. We show that taking *any* minimal FES already yields a 3 approximation to the VFES problem. We then show that a local search method gives an approximation factor of $2 + \epsilon$ for any fixed $\epsilon > 0$ for the minimum VFES problem. However, the running time of this algorithm is $O(n^{O(1/\epsilon)})$. Since this algorithm is not practical, we develop a restricted local search algorithm that restricts the local search neighborhood, which yields a significantly improved running time of $O(n^3 + n^2 f(\frac{1}{\epsilon}))$, and actually yields a practical algorithm with the same approximation guarantee. (Here f is an exponential function but has no dependence on n.) We also show that, for planar graphs, we can design a PTAS. In other words, for any fixed $\epsilon > 0$, we obtain a $1 + \epsilon$ approximation in polynomial time.

3. Hardness results.

THEOREM 3.1. *The minimum VFES problem is MAX SNP-hard and NP-hard.*

Proof. In Lemma 3.2, we show that there is a polynomial time reduction from the independent set problem to the complement of the minimum VFES problem, namely, the FDST problem, such that there is an independent set of size I in a graph G with n nodes and m edges, if and only if there is an FDST solution of size $I + 1$ in a graph H with $N = m + n + 3$ nodes. This implies that G has a vertex cover of size VC if and only if H has a solution to the minimum VFES problem of size $N - (n - VC + 1) = m + VC + 2$.

The vertex cover problem is MAX SNP-hard for graphs in which all degrees are bounded by 3 [1]. Let G be one such graph. Then any vertex cover of G has size $VC \geq m/3$. Let VC^* be the size of the optimal vertex cover of G, and let OPT be the size of the optimal solution to the minimum VFES problem on H. Similarly, let VC and $VFES$ be the size of a vertex cover of G and the size of the corresponding

minimum VFES on H respectively. Then it follows (assuming $m > 0$) that

$$OPT \leq m + VC^* + 2 \leq 4VC^* + 2 \leq 6VC^*.$$

Also,

$$|VFES - OPT| = |m + VC + 2 - (m + VC^* + 2)| = |VC - VC^*|.$$

Thus a $1+\epsilon$ approximation algorithm for the minimum VFES can be used to construct a $1+6\epsilon$ approximation algorithm for the bounded degree vertex cover problem. Hence the minimum VFES problem is MAX SNP-hard. □

The following lemma is based on the NP-completeness proof of the FDST problem [10].

LEMMA 3.2. *There is a polynomial time reduction from the independent set problem on a graph G with n nodes and m edges to the FDST problem on a graph H with $m + n + 3$ nodes such that there is an independent set of size I in G if and only if there is an FDST of size $I + 1$ in H.*

Proof. Given a graph G, an input instance of the independent set problem (we will assume without loss of generality that G has at least two edges, has no isolated vertices, and has a maximum independent set of size at least 3), we create an instance of the FDST problem as follows. Graph H can be viewed as a four-layer graph whose edges connect only vertices in adjacent layers. Hence H is bipartite. Layer one consists of just one vertex a. Layer two has one vertex for every vertex of G, and every vertex in the second layer is connected to a. Layer three has one vertex for every edge of G, and if (u, v) is an edge in G, then the corresponding vertex in the third layer is connected to the vertices in layer two, corresponding to the vertices u and v of G. Finally, layer four has two vertices b and c, which are both connected to every vertex in the third layer.

Let T be a feasible solution to the FDST problem for graph H. First, note that if any two vertices have at least two common neighbors in H, then they both cannot have full degree in T. Hence only one vertex from among $\{b, c\}$ might have full degree in T. This is because G has at least two edges, and hence b and c have at least two common neighbors. Similarly, at most one vertex in the third layer of H has full degree in T. This is because both b and c are in the neighborhood of every vertex in the third layer of H. If vertex a has full degree in T, then none of the vertices in the third layer of H have full degree in T. Vertex a and any vertex in layer three have two common neighbors in layer two. Finally, note that all vertices in the second layer with full degree in T must form an independent set in G.

The above implies that, if G has an independent set of size I, then there is a feasible solution of size $I + 1$ to the FDST problem for the graph H. In this solution, vertex a and the vertices in the independent set have full degree. Similarly, if there is a feasible solution to the FDST problem for the graph H of size $I+1$, then if b or c have full degree (only one of them can be a full degree vertex), then no pair of vertices in the second layer can have full degree. To see this, suppose that there are two vertices in the second layer that have full degree. Note that each one of them has neighbors in layer three since there are no isolated vertices in G. They also have edges to a. By the assumption that one of b or c has full degree, this would thus imply the presence of a cycle in the solution to the FDST, which is a contradiction. We cannot pick more than one full degree vertex in layer three in any case, and we also can only pick vertex a or a vertex in layer three of full degree but not both. Therefore, we will be able to pick at most three vertices of full degree. Hence at least I vertices in layer two

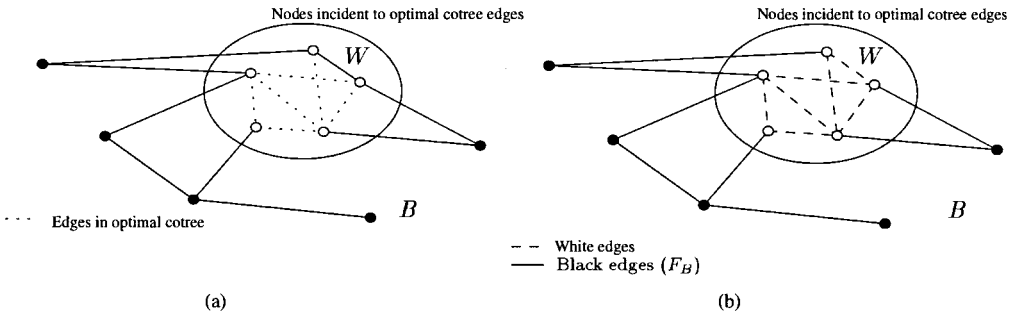


FIG. 4.1. *Example showing black and white vertices.*

have full degree in this feasible solution and therefore G has an independent set of size I. □

4. The minimum VFES problem. In this section, we consider the minimum VFES problem, namely, to find an FES incident on the minimum number of vertices. Any minimal FES is also a cotree.

We show that iteratively deleting any edge from a cycle in the graph $G = (V, E)$ yields an approximation algorithm with a performance ratio of 3. We then improve the solution by applying a "local-improvement" strategy and prove that this is a $2+\epsilon$ approximation for any fixed $\epsilon > 0$. We will also assume that our input graph is connected.

Let S be any subset of edges. Let $V(S)$ be the subset of vertices in G incident on edges in S. Let C_{OPT} be the edges of a minimum VFES (cotree). Let W (white vertices) be $V(C_{OPT})$. Let B (black vertices) be the set of remaining vertices, $V \setminus W$. Edges of G incident to at least one black vertex are colored black and all remaining edges are colored white. (See Figure 4.1(a) for a minimum cotree and Figure 4.1(b) for the colors of edges.) The set of black edges induces a forest F_B in G. We refer to the set of white edges as E_W.

We first prove the following theorem about minimal FESs.

THEOREM 4.1. *Any minimal FES gives a* 3 *approximation to the VFES problem.*

Proof. Let S be a minimal FES. Let $S_w = S \cap E_W$ and $S_b = S \setminus S_w$. We prove that $|V(S)| \leq 3|V(C_{OPT})| - 2$ as follows. Since $|V(E_W)| = |W| = |V(C_{OPT})|$, we have $|V(S_w)| \leq |V(C_{OPT})|$. The proof of the following inequality is given in Lemma 4.2:

$$|S_b| \leq |V(C_{OPT})| - 1.$$

Since the number of vertices that can be incident on any set of edges is at most twice the number of edges, we conclude that

$$|V(S_b)| \leq 2(|V(C_{OPT})| - 1).$$

Thus $|V(S)| \leq |V(S_w)| + |V(S_b)| \leq 3|V(C_{OPT})| - 2$. □

LEMMA 4.2. *Let S be a minimal FES and S_b the set of black edges in this set; then*

$$|S_b| \leq |V(C_{OPT})| - 1.$$

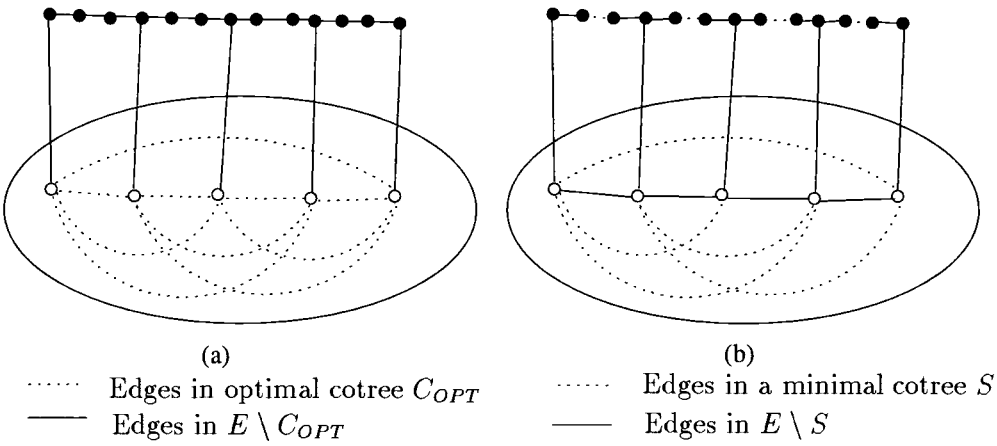


FIG. 4.2. *Example to show that factor* 3 *bound is tight.*

Proof. Recall that the black edges form a forest F_B. Any black edge $e \in S_b$ is essential to a minimal FES. In other words, the introduction of edge e into $E \setminus S$ would create a cycle C_e. C_e contains only edges in $E \setminus S$, except for edge e. Since every cycle in G has at least one white edge, C_e must also have two white vertices incident to this white edge. Walking around the cycle from e in both directions, we stop when we first encounter a white vertex. (We will stop at two distinct white vertices.) Consider $F_B \setminus S_b$; edge e connects two components, each of which must contain at least one white vertex. If $|S_b| \geq |V(C_{OPT})|$, then there are at least $|V(C_{OPT})| + 1$ distinct components in $F_B \setminus S_b$. In the forest F_B, there are exactly $|V(C_{OPT})|$ white vertices; thus one edge in S_b connects two components such that one component contains no white vertex, which is a contradiction. □

It is possible to make examples that show that this bound is tight (see Figure 4.2). If there are k vertices in the optimal cotree, then a minimal solution may have up to $k + 2(k-1)$ vertices.

4.1. Local improvement. We can apply the following local-improvement strategy to our minimal FES S. Let k be $\lceil \frac{1}{\epsilon} \rceil$. We will show that we can prove an approximation bound of $2 + \frac{1}{k}$ using a simple local search algorithm. Replace a set of at most $k-1$ edges from S to get another FES with lower cost (cost is the number of vertices incident to the edges in the FES). Keep doing this until no reduction in cost is possible. This new solution will be referred to as S^k. This can be done for any fixed $k > 0$. (For $k = 1$, this is simply any minimal FES.)

We call this the k-Local-Improvement Algorithm.

k-Local Improvement(G)—

1. Pick an arbitrary cotree S (of size $|E| - (|V| - 1)$).
2. **while** there exists a cotree S' such that
   $|V(S')| < |V(S)|$ and $|S \bigcap S'| > |S| - k$ **do**
3. $S \leftarrow S'$.
4. $S^k \leftarrow S$.
5. **end**

We can partition the solution S^k obtained by the algorithm into two sets as

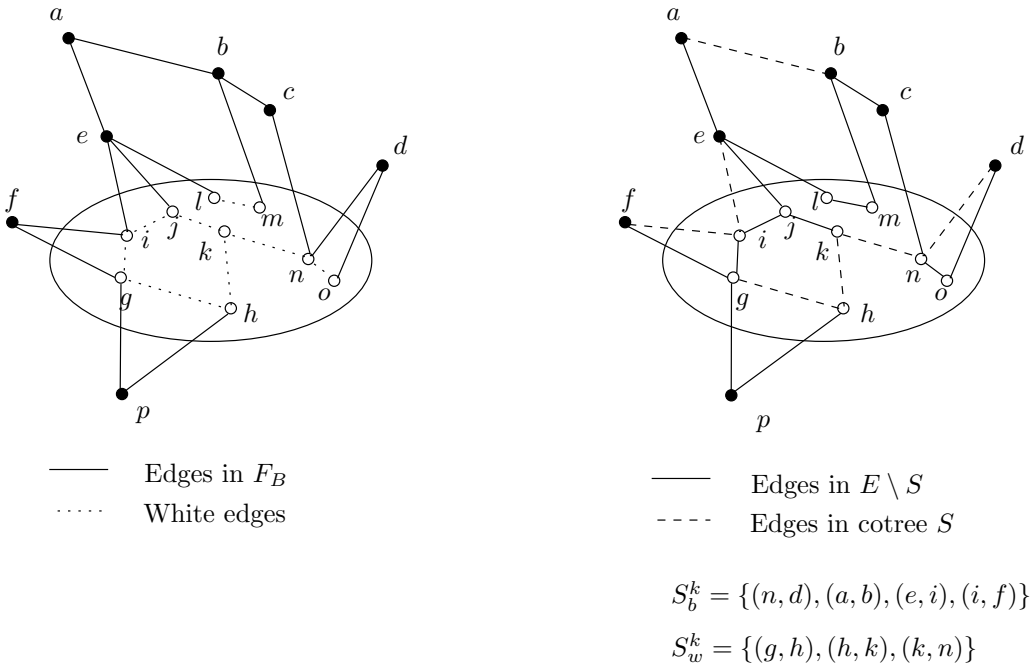


FIG. 4.3. *Figure to show full and nonfull black components.*

before—white edges S_w^k and black edges S_b^k. The edges in S_b^k form a collection of connected components. Let $\mathcal{C} = \{C_i | C_i \text{ is a component of black edges from } S_b^k\}$.

For each black component C_i that shares at least one white vertex with S_w^k, $|V_b(C_i)| \leq |E(C_i)|$, where $V_b(C_i)$ denotes the black vertices of C_i. A black component that does not share any vertices with S_w^k is called a *full black component*. In Figure 4.3, we show three black components: $\{a, b\}$, $\{e, f, i\}$, and $\{d, n\}$. The first two components are full black components. The third component shares a white vertex (n) with an edge in S_w^k.

A full black component may have all black vertices or may have white vertices in $W \setminus V(S_w^k)$. We call a full black component C_i *small* if it has at most $k-1$ edges. The other full black components in $\mathcal{C}$ are referred to as *large*. $\mathcal{C}_{\text{small}}$ is the set of small full black components. $\mathcal{C}_{\text{large}}$ is the set of large full black components. The remaining components are referred to as $\mathcal{C}_{\text{nonfull}}$.

The proof of the main theorem relies on a lemma bounding the number of small full black components $|\mathcal{C}_{\text{small}}|$ by $|W| - |V(S_w^k)|$ (Lemma 4.6). We will prove this lemma next.

Root each black tree in F_B at an arbitrary black vertex, and orient all edges away from the root (see Figure 4.4). Let C_i be a small (full) black component in S^k rooted at r_i.

Number the vertices in C_i as $u_1^i, \ldots, u_{p_i+1}^i$. Let e be the (oriented) edge (u_j^i, u_k^i). Let Shadow(i, e) be the set of white vertices reachable from u_k^i by following a directed path of zero or more black edges that are in $F_B \setminus S^k$. The vertices in Shadow(i, e) are descendants of u_k^i in the rooted black forest F_B.

Let Shadow$(C_i) = \cup_{e \in C_i}$Shadow(i, e). Note that Shadow(C_i) is the set of white vertices that are reachable from vertices in $C_i - \{r_i\}$ by following a directed path of

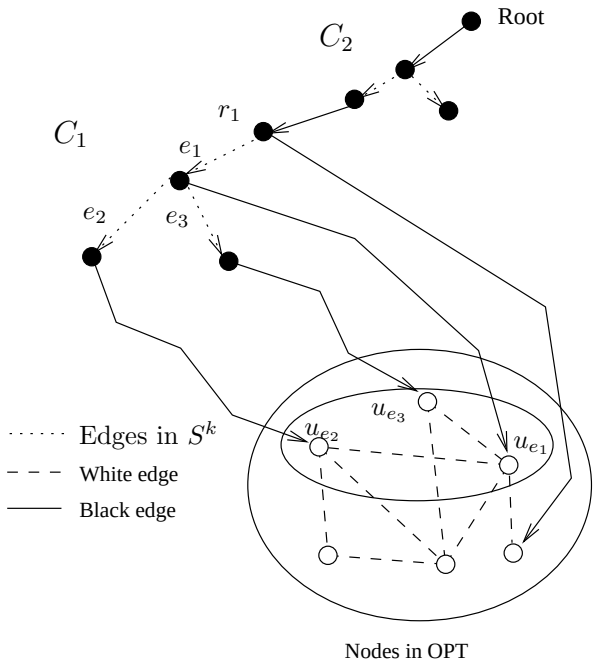


FIG. 4.4. *Example showing black components.*

black edges that are in $F_B \setminus S^k$.

LEMMA 4.3. *For any full black component C_i and $e \in C_i$, we have Shadow$(i, e) \neq \emptyset$.*

Proof. Consider an (oriented) edge $e = (u_j^i, u_k^i) \in C_i$. If $u_k^i \in W$, then clearly Shadow$(i, e) \neq \emptyset$. Now assume that $u_k^i \in B$. If we add edge e to $E \setminus S^k$, we get a cycle C_e with a white edge on it connecting two white vertices. Note that all the edges in F_B incident on u_k^i except for the edge from the parent u_j^i are oriented away from u_k^i. Consider the first white vertex on the path in C_e from node u_k^i to u_j^i avoiding e; we take edges from $F_B \setminus S^k$ before reaching this white vertex. Thus there is a directed path of zero or more black edges from u_k^i using edges from $F_B \setminus S^k$ to some white vertex. □

LEMMA 4.4. *For each pair of black components C_i and C_j, Shadow(C_i) and Shadow(C_j) are disjoint.*

Proof. Suppose (for the sake of contradiction) that $w \in \text{Shadow}(C_i) \bigcap \text{Shadow}(C_j)$. By the definition of Shadow, $\exists u \in C_i - \{r_i\}$ (and $\exists v \in C_j - \{r_j\}$) such that there is a directed path of black edges in $F_B \setminus S^k$ from u (and v) to w. Since w is a descendant of both u and v in the black tree F_B, one of $\{u, v\}$ is a descendant of the other. Assume that u is an ancestor of v. The unique path from u to w in F_B includes the tree path from u to v and the path from v to w. Since the edge joining v to its parent is in C_j and in S^k, at least one edge along this path is not in the set $F_B \setminus S^k$, and thus w is not in the shadow of u. □

Consider a small (full) black component C_i with $p_i (< k)$ edges.

LEMMA 4.5. *For each small (full) black component C_i, Shadow(C_i) contains at least one vertex in $W \setminus V(S_w^k)$.*

Proof. Assume for the sake of contradiction that all the vertices in Shadow(C_i) are

in $V(S_w^k)$. This means that each of these nodes is incident to at least one white edge in S^k. Adding a new edge to the cotree that is incident on a vertex in Shadow(C_i) can only increase the cost of the solution by at most one since one end point of such an edge is already incident to some white edge in the cotree. Let T_{S^k} be the tree corresponding to $E \setminus S^k$. We call R_i a *replacement set* for C_i if $T_{S^k} \cup C_i \setminus R_i$ is a spanning tree in G. We now show that we can find a valid replacement set R_i for C_i of cost at most p_i. By this replacement, we will obtain a strictly better solution since C_i is incident to $p_i + 1$ vertices, and R_i has cost p_i. This exchange involves a set of at most $k-1$ edges, contradicting the assumption that we have a solution that cannot be locally improved.

We now show how to find the set R_i. Note that the solution S^k is minimal, and the introduction of any edge from S^k into the tree formed by $E \setminus S^k$ will create a cycle.

Initially, $R_i = \emptyset$ and $T = T_{S^k}$. As we add edges from C_i to T, we will delete edges from T to maintain a tree. These deleted edges will be added to R_i to form the replacement set for C_i. We process the edges of C_i as we encounter them in a preorder traversal of C_i from r_i. Assume that we are processing the edge $e = (u_j^i, u_k^i) \in C_i$ (oriented downward (away from the root)). If we add the edge e to T, we get a cycle C_e. We follow the path in C_e starting from u_k^i and going away from u_k^i and stop at the first white edge (u_e, v_e). By Lemma 4.3, such an edge will be encountered. If $u_e \in C_i$, then this cannot be a full black component since $u_e \in V(S_w^k)$, and we obtain a contradiction. Since $u_e \in$ Shadow(i, e), it is in $V(S_w^k)$ and does not cost anything. The edge (u_e, v_e) is the replacement edge for e and costs at most one. We add e to T and remove (u_e, v_e) from T and add (u_e, v_e) to R_i. We repeat this until we find all the replacement edges for C_i. ∎

LEMMA 4.6. *The number of small (full) black components in S^k is at most $|W| - |V(S_w^k)|$.*

Proof. We charge each small black component C_i to a vertex in $W \setminus V(S_w^k)$ that also belongs to Shadow(C_i). Since the shadows are disjoint (Lemma 4.4), each node in $W \setminus V(S_w^k)$ is charged at most once, and the lemma follows. ∎

THEOREM 4.7. *The k-Local-Improvement Algorithm obtains a solution that is at most $2 + \frac{1}{k}$ times the optimal solution. This algorithm runs in polynomial time for any fixed $k > 0$.*

Proof. We bound the number of small full black components $|\mathcal{C}_{\text{small}}|$ by $|W| - |V(S_w^k)|$ (Lemma 4.6). For each large full black component C_i, the number of vertices $|V(C_i)| = |E(C_i)| + 1 \leq |E(C_i)| + \frac{E(C_i)}{k} \leq \frac{k+1}{k}|E(C_i)|$.

For each black component with at least one white vertex also in $V(S_w^k)$, we have $|V_b(C_i)| \leq |E(C_i)|$.

$$
\begin{aligned}
|V(S_b^k) \setminus V(S_w^k)| &= \sum_{C_i \in \mathcal{C}_{\text{small}}} |V(C_i)| + \sum_{C_i \in \mathcal{C}_{\text{large}}} |V(C_i)| + \sum_{C_i \in \mathcal{C}_{\text{nonfull}}} |V_b(C_i)| \\
&\leq \sum_{C_i \in \mathcal{C}_{\text{small}}} (|E(C_i)| + 1) + \sum_{C_i \in \mathcal{C}_{\text{large}}} \frac{k+1}{k} |E(C_i)| + \sum_{C_i \in \mathcal{C}_{\text{nonfull}}} |E(C_i)| \\
&\leq |\mathcal{C}_{\text{small}}| + \sum_{C_i \in \mathcal{C}_{\text{small}}} |E(C_i)| + \sum_{C_i \in \mathcal{C}_{\text{large}}} \frac{k+1}{k} |E(C_i)| \\
&\qquad + \sum_{C_i \in \mathcal{C}_{\text{nonfull}}} |E(C_i)|.
\end{aligned}
$$

By Lemma 4.6, we get

$$
|V(S_b^k) \setminus V(S_w^k)| \leq |W| - |V(S_w^k)| + \sum_{C_i \in \mathcal{C}} \frac{k+1}{k} |E(C_i)|.
$$

Note that

$$
|V(S^k)| = |V(S_w^k)| + |V(S_b^k) \setminus V(S_w^k)|.
$$

Using the bound on $|V(S_b^k) \setminus V(S_w^k)|$ obtained above, we get

$$
|V(S^k)| \leq |V(S_w^k)| + |W| - |V(S_w^k)| + \frac{k+1}{k} \sum_{C_i \in \mathcal{C}} |E(C_i)|.
$$

By Lemma 4.2, since S_b^k is also a subset of a minimal cotree, we can bound $\sum_{C_i \in \mathcal{C}} |E(C_i)|$ by $|V(C_{OPT})| - 1$.

$$
\begin{aligned}
|V(S^k)| &\leq |W| + \frac{k+1}{k}(|V(C_{OPT})| - 1) \\
&= |V(C_{OPT})| + \frac{k+1}{k}(|V(C_{OPT})| - 1).
\end{aligned}
$$

This gives us a $2 + \frac{1}{k}$ approximation to VFES.

A naive bound on the running time of the algorithm can be computed as follows. There are at most n iterations since in each iteration our cost decreases by at least one. To implement an iteration, we have to try subsets of at most $k-1$ edges from our cotree and try to replace them by another subset of the same size such that we obtain a cotree with lower cost. An easy bound on the running time for a single iteration is $O(m^{k-1}n^k)$. □

Since this algorithm is not very practical, we show how to modify the algorithm to obtain an efficient implementation.

4.2. Restricted local improvement. We show that we can obtain the $2 + \frac{1}{k}$ bound by modifying the algorithm to improve only "small trees." In other words, once we have a cotree S (corresponding to tree T), we can consider its connected components and try to reduce $|V(S)|$ by eliminating only small trees (of size $< k$) from S. We check to see if we can delete the edges of a small tree T_s from S (this will create cycles in $T \cup T_s$) and then find a set of edges to add to S so that we again

get a cotree. If this replacement reduces the cost of the cotree, we perform this step. Keep doing this until either no small trees are left or we cannot get a reduction in the number of vertices that are incident to the edges in the cotree.

THEOREM 4.8. *The Restricted-Local-Improvement Algorithm obtains a solution that is at most* $2 + \frac{1}{k}$ *times the optimal solution.*

Proof. The proof of this theorem is essentially the same as the proof of the k-Local-Improvement Algorithm. Basically, we observe that the only place where we actually use any specific property of the k-Local-Improvement Algorithm is in the proof of Lemma 4.5. This proof assumes only that components of the solution that are trees are chosen as candidates to be eliminated during the local search. Hence the same bound as in Theorem 4.7 follows. □

This method can be implemented in $O(n^{k+2})$ time as follows: We show that each iteration can be implemented in $O(n^{k+1})$ time, and there are at most n iterations. There are only $O(n)$ possible "small trees" to eliminate. For each small tree, we have to find a valid replacement set—this takes $O(n^k)$ time since there are $O(n^{k-1})$ possible replacement steps, and checking each one takes $O(n)$ time. Thus this gives a bound of $O(n^{k+1})$ for each iteration. The cost is measured as the number of vertices incident to the set of edges in our subset of edges. The initial cost of the cotree is at most n, the number of vertices. Since in each iteration this cost decreases by at least one, we have at most n iterations (there could be far fewer in practice) giving the overall bound of $O(n^{k+2})$.

The key difficulty is in implementing the replacement step efficiently. The naive implementation takes $O(n^{k+1})$ time as we just observed. We now give an algorithm whose complexity (per iteration) is $O(n^2 + nf(k))$, where $f(k)$ is an exponential function in k. The key point is that this algorithm is practical, whereas the previous one is not practical. This improvement is obtained by showing that we need only to "search" for an improvement by restricting our search space to a very small number of possibilities. This gives an overall bound of $O(n^3 + n^2 f(k))$ for the entire algorithm.

We now focus on the implementation of a single iteration of the algorithm. Suppose we have a current cotree S, and let T be the tree $E \setminus S$. Let T_s be a tree with $\ell \leq k-1$ edges in the graph (V, S). Our goal is to try to find a "replacement" set of ℓ edges R such that $(S \setminus T_s) \cup R$ is a cotree or, equivalently, $(T \cup T_s) \setminus R$ is a tree. If the cost of $(S \setminus T_s) \cup R$ is lower than the cost of S, we obtain a local improvement (reduction in cost).

Each edge $e \in T_s$ induces a cycle in T. Path $P(T, e)$ is obtained by removing e from this cycle. Let $\mathcal{T}$ be the subtree of T defined by taking the union of the ℓ paths $P(T, e)$ for each $e \in T_s$. Clearly, $R \subseteq \mathcal{T}$. Figure 4.5 shows the tree T along with a tree $T_s \subseteq S$ (of three edges in this case) and the corresponding paths $P(T, e)$. Our goal is to find a replacement set R of lower cost. Observe that, since we need to "destroy" the ℓ cycles that are created when we add the ℓ edges of T_s to T, we are restricted to picking edges from $\mathcal{T}$ to obtain the set R. We now show that, even though $\mathcal{T}$ can be large, we can find a subtree $\mathcal{T}'$ of size $O(\ell)$ such that if a replacement set R of lower cost exists, then a replacement set of edges chosen from $\mathcal{T}'$ can be found.

We now define tree $\mathcal{T}'$. Consider $T_s \cup \mathcal{T}$. Mark the vertices of T_s as "special" and all the vertices in $\mathcal{T} \setminus T_s$ with degree three or more in the tree $\mathcal{T}$ as "special." (Special vertices are shown as the circled vertices in Figure 4.5.) We can now think of $\mathcal{T}$ as a "tree-like" structure on "special" vertices, where we have paths connecting the "special" vertices. If T_s has ℓ edges, then there are at most 2ℓ special vertices, and thus there are at most $2\ell - 1$ "paths" in $\mathcal{T}$ connecting the special vertices. (This

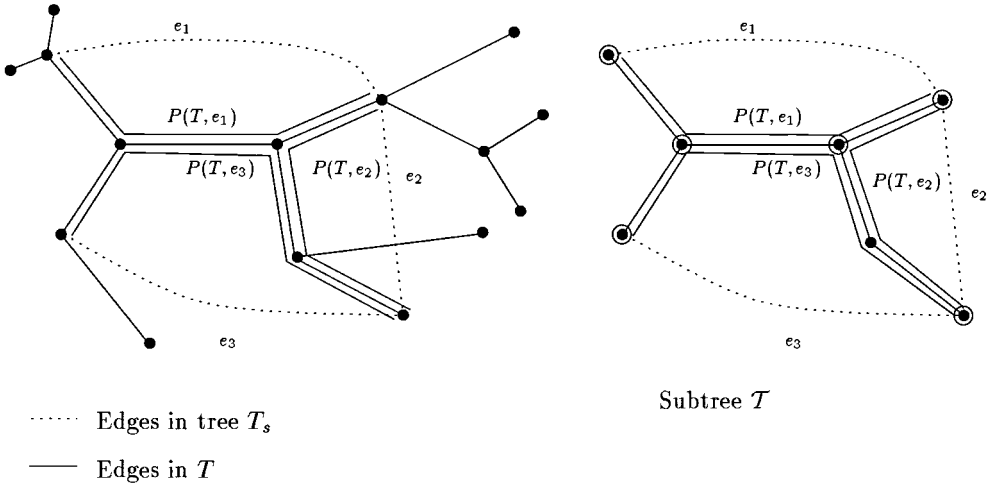


FIG. 4.5. *Figure to show* $\mathcal{T}$.

will be proven formally in Lemma 4.10.) If any "path" between two special vertices has five or more edges, then we will "shrink" this to a path of length at most four. (This will be shown later.) This will ensure that each path has a constant number of edges, and thus the size of the entire tree $\mathcal{T}'$ is $O(\ell)$. Each "path" between two special vertices is also referred to as a "link."

The cost of the ℓ edges in T_s is exactly $\ell + 1$. The vertices in $V(S \setminus T_s)$ are called *marked*. All other vertices are *unmarked*. Our goal is to pick a replacement set R of cost at most ℓ. Our replacement set of edges is incident on marked and unmarked vertices, but only the unmarked vertices cost 1, while the marked vertices cost 0 since these nodes are already incident on edges in $S \setminus T_s$. To do this, we will enumerate over subsets of size ℓ from $\mathcal{T}'$. For each candidate set, we will check if it is a valid replacement set and compute the cost of the new cotree. If the cost of the new cotree is lower, we perform the replacement. It should now be clear that the function f is the number of ways of choosing ℓ edges from the $2\ell - 1$ links so that they form a valid replacement set.

LEMMA 4.9. *If a replacement set R of cost lower than the cost of T_s exists, then there exists a replacement set of edges chosen from $\mathcal{T}'$ of the same cost as R.*

Proof. We first make some observations about the replacement set R. In the replacement set R, at most one edge can be chosen from each link. Let the path $P = [v_0, \ldots, v_k]$ (see Figure 4.6) be the path corresponding to a link. If $k \leq 4$, we do not change the link. (This link has at most four edges.) If $k > 4$, we will define a new compressed link for $\mathcal{T}'$ based on the subset of marked vertices on path P. If there is a replacement set R in $\mathcal{T}$, then picking the corresponding edge from $\mathcal{T}'$ yields a replacement set of the same cost.

If there is no marked vertex on P and R chooses an edge from P, then, without loss of generality, R picks an edge incident to v_0 or v_k. We can compress this path to a path with two edges.

If there exists a pair of adjacent marked vertices on P and R picks an edge from P, then, without loss of generality, we may assume that R picks the edge between a pair of marked vertices. This can be shown by arguing that, if R does not pick an

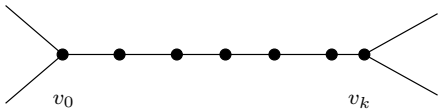


FIG. 4.6. *A long path that will be compressed.*

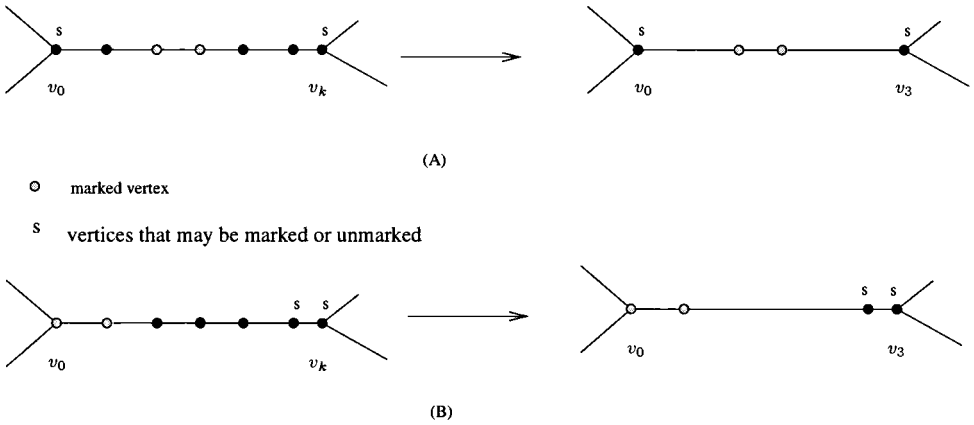


FIG. 4.7. *Compressing a link when there is an adjacent pair of marked vertices.*

adjacent pair of marked vertices, then we can change R to another valid replacement set which uses the adjacent marked pair without increasing its cost. See Figure 4.7 to see the "compression" of a long path to a short path. In case (A), the marked pair is an internal pair on the path, and, in case (B), it is an end pair on the path. In either case, we perform the transformation as shown in Figure 4.7. We leave it to the reader to see that a replacement set of edges in $\mathcal{T}$ can be translated to a replacement set of edges in $\mathcal{T}'$ with no increase in cost.

If there exists at least one marked vertex on P but no adjacent pair of marked vertices and R chooses an edge from P, then, without loss of generality, R picks an edge that is incident on a marked vertex. In each case, it can be verified that we preserve the cost of the set R in $\mathcal{T}$ when we consider the corresponding replacement set R' in $\mathcal{T}'$. Figure 4.8 shows how we compress the link. In each case, it can be verified that we preserve the cost of the set R in $\mathcal{T}$ when we consider the corresponding replacement set R' in $\mathcal{T}'$. We consider all cases based on the status of the vertices marked "s." We do not show the symmetric cases and the cases when an adjacent marked pair exists. The proof for each case is left to the reader. □

LEMMA 4.10. *The size of $\mathcal{T}'$ is $O(\ell)$.*

Proof. In $\mathcal{T}'$ there are at most 2ℓ special vertices. This can easily be proven by induction on the number of edges in T_s. If T_s has one edge, then it creates exactly two marked vertices. Consider T_s, and delete an edge incident to a leaf vertex of T_s. This has $\ell - 1$ edges and thus $2(\ell - 1)$ special vertices. When we add the leaf edge, we create one more special vertex (the leaf vertex) and possibly one more vertex whose degree becomes three.

Thus in $\mathcal{T}$ there are at most $2\ell - 1$ "links," where a "link" is a maximal sequence of degree two vertices. Each link is a sequence of at most four edges, hence the number of edges in $\mathcal{T}'$ is at most $4(2\ell - 1)$, which is $O(\ell)$. □

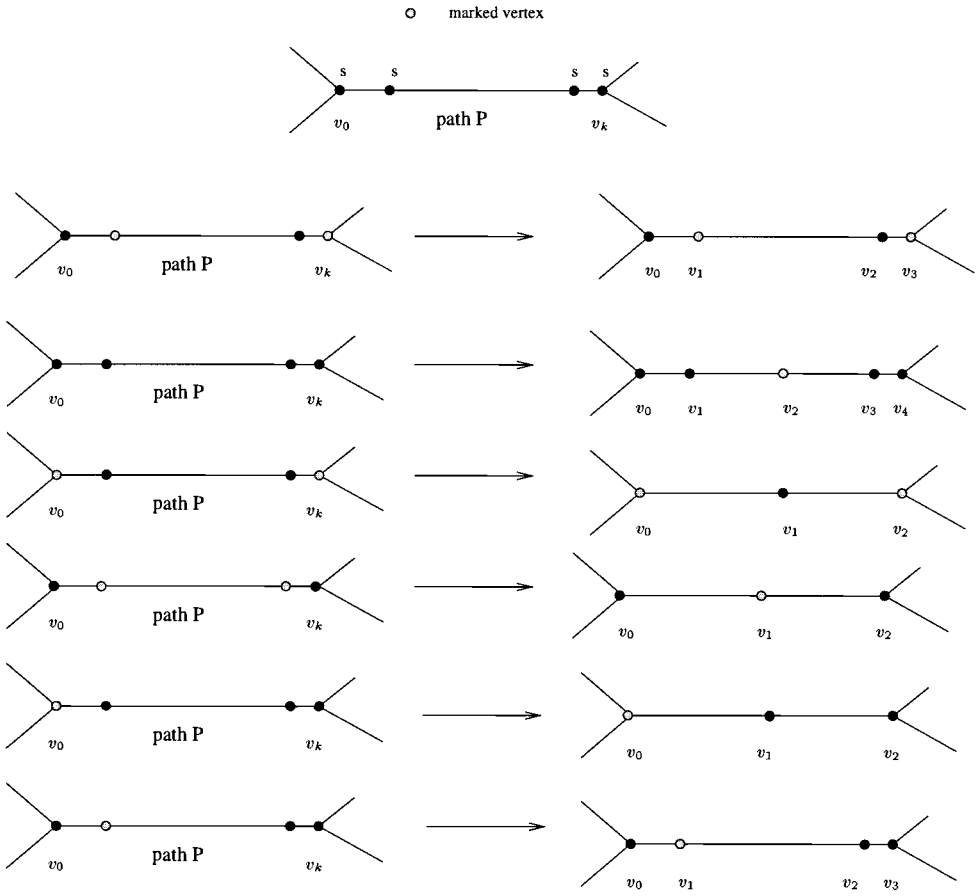


FIG. 4.8. *Compressing a link when there is no adjacent pair of marked vertices.*

5. Planar graphs. In this section, we give a $(1 + \epsilon)$ approximation for the VFES problem for any fixed $\epsilon > 0$ (also referred to as a PTAS). A PTAS for the VFES problem requires two steps. First, we preprocess the graph to compute its 2-edge connected components. We can work with each 2-edge connected component separately since the edges that are cut-edges (bridges) cannot be on any cycle. We then preprocess the graph to eliminate adjacent pairs of degree two vertices and then prove that, in the resulting graph, the optimal solution has size $\Omega(n')$, where n' is the number of vertices, after the elimination of adjacent pairs of vertices with degree two. We then explain the necessary modifications to Baker's schema for generating a PTAS by using optimal solutions for k-outerplanar graphs [4]. Baker's schema requires an optimal solution to a version of the VFES problem when restricted to k-outerplanar graphs; this is assumed to be available as a subroutine here and is presented explicitly in section 6.

We first show that the optimal solution has size $\Omega(n')$. Observe that a maximal

sequence of degree two vertices can be contracted into a sequence containing only one degree two vertex without changing the cost of the optimal solution. We then prove that the number of edges in the graph is at least $\frac{6}{5}n'$, and hence any FES must have at least $\frac{n'}{5}$ edges.

LEMMA 5.1. *If G is a 2-edge connected graph with n' vertices and has no adjacent vertices of degree 2, then the number of edges is at least $\frac{6}{5}n'$.*

Proof. Let n_i be the number of vertices of degree i. Since the graph is 2-edge connected, $n_1 = 0$. (Each vertex has degree at least 2.) Since there is no edge between two vertices of degree 2, we have $|E| \geq 2n_2$. If $n_2 \geq \frac{3}{5}n'$, then we immediately obtain $|E| \geq \frac{6}{5}n'$. Hence we assume that $n_2 \leq \frac{3}{5}n'$. Note that $2|E| = \sum_{i=2}^{n-1} in_i$. Hence $|E| = \sum_{i=2}^{n-1} \frac{i}{2}n_i$. Since $\sum_{i=3}^{n-1} n_i = n' - n_2$, $|E| \geq n_2 + \sum_{i=3}^{n-1} \frac{3}{2}n_i = n_2 + \frac{3}{2}(n' - n_2)$. We obtain $|E| \geq \frac{3}{2}n' - \frac{1}{2}n_2 \geq \frac{6}{5}n'$. ∎

Baker gives a framework which constructs a PTAS using optimal solutions for k-outerplanar graphs—planar graphs where all nodes have a path of length less than or equal to k to a node on the outermost face [4].

Fix a planar embedding of the graph. Let $d(v)$ equal the shortest path length from v to any node on the outer face of G. This scheme creates a collection of k decompositions of the planar graph G into a set of k-outerplanar graphs. For each value of $i = 0 \ldots (k-1)$, we generate the ith decomposition as follows: delete edges that connect nodes with label $d(v) - 1$ and $d(v)$, where $d(v)$ is congruent to $i (\mathrm{mod}\ k)$. For example, for $i = 0$, we delete edges connecting nodes with $d(v)$ value $k - 1$ and k, $2k - 1$ and $2k$, etc. After we delete these edges, we are left with a graph G_i, which is a collection of connected components that are each k-outerplanar. We obtain the optimal solution for G_i by running a linear time algorithm for each connected component (see section 6). This optimal VFES for the graph G_i can be extended to become an FES for the input graph G by including a subset of the edges deleted earlier, with an additional cost less than or equal to the number of nodes with label $d(v) - 1$ and $d(v)$, where $d(v)$ is congruent to $i (\mathrm{mod}\ k)$. Define the set of nodes with these labels to be R_i; then the cost of our solution is at most $|OPT| + |R_i|$.

A given partition of G into a collection of k-outerplanar graphs G_i will have a solution of size at most $|OPT| + |R_i|$, where R_i is the set of "boundary" nodes in G_i. If, for all $i, |R_i| > \frac{2n'}{k}$, then $\sum_{i\text{even}} |R_i| > n'$, which is a contradiction. Therefore, for some i, we obtain a solution of size at most $|OPT| + \frac{2n'}{k}$.

By Lemma 5.1, the optimal FES must have at least $\frac{n'}{5}$ edges. Since the graph is planar, these edges must be incident on $\Omega(n')$ vertices; $|OPT| \geq cn'$ for some constant c. Our solution cost is at most $OPT + \frac{2OPT}{kc} \leq OPT(1 + \frac{2}{kc})$. By setting $k = \frac{2}{c\epsilon}$, we get a bound of $(1 + \epsilon)OPT$.

6. *k*-outerplanar and treewidth-bounded graphs. In this section, we give a linear time algorithm for treewidth-bounded graphs. (If the graph has treewidth k, then the time required for the algorithm to run will be exponential in k but linear in the size of the graph.) Bodlaender [7] proves that any k-outerplanar graph has treewidth at most $3k - 1$. Since our graph G_i consists of a collection of connected components, each of treewidth $3k - 1$, we can run this algorithm on each component separately and take the union of the solutions we obtain. These algorithms rely on a tree-decomposition of a graph, which we define for completeness.

DEFINITION 6.1. *Let $G = (V, E)$ be a graph. A* tree-decomposition *of G is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, where $\{X_i \mid i \in I\}$ is a family of subsets of V and $T = (I, F)$ is a tree with the following properties:*

1. $\bigcup_{i \in I} X_i = V$.
2. *For every edge* $e = (v, w) \in E$, *there is a subset* X_i, $i \in I$, *with* $v \in X_i$ *and* $w \in X_i$.
3. *For all* $i, j, k \in I$, *if* j *lies on the path from* i *to* k *in* T, *then* $X_i \bigcap X_k \subseteq X_j$.

The treewidth *of a tree-decomposition* $(\{X_i \mid i \in I\}, T)$ *is* $\max_{i \in I}\{|X_i| - 1\}$. *The* treewidth *of a graph is the smallest value* k *such that the graph has a tree-decomposition with treewidth* k.

Many problems are known to have linear time algorithms on graphs with constant treewidth, and there are frameworks for automatically generating a linear time algorithm, given a problem specification in a particular format [2, 8]. The VFES problem can be expressed in the formalism of Borie, Parker, and Tovey [8] as follows: given input graph $G = (V, E)$, find $\min |V_1| [Forest(V, E \setminus E'), Inc(E', V_1)]$, which states that we want to minimize the set of nodes incident upon a set of edges whose removal makes the graph a forest. $Forest(V, E \setminus E')$ and $Inc(E', V_1)$ are predicates that specify that the edges induced by the set $E \setminus E'$ form a forest and V_1 is the set of vertices incident on the edges in E'.

Acknowledgments. We thank Sudipto Guha, Yoram Sussmann, and An Zhu for useful discussions. We thank the reviewer for comments.

REFERENCES

[1] P. Alimonti and V. Kann, *Some APX-completeness results for cubic graphs*, Theoret. Comput. Sci., 237 (2000), pp. 123–134.
[2] S. Arnborg, J. Lagergren, and D. Seese, *Easy problems for tree-decomposable graphs*, J. Algorithms, 12 (1991), pp. 308–340.
[3] V. Bafna, P. Berman, and T. Fujito, *A* 2*-approximation algorithm for the undirected feedback vertex set problem*, SIAM J. Discrete Math., 12 (1999), pp. 289–297.
[4] B. Baker, *Approximation algorithms for NP-complete problems on planar graphs*, J. ACM, 41 (1994), pp. 153–190.
[5] R. Bar-Yehuda, D. Geiger, J. (S.) Naor, and R. M. Roth, *Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference*, SIAM J. Comput., 27 (1998), pp. 942–959.
[6] A. Becker and D. Geiger, *Approximation algorithms for the loop cutset problem*, in Proceedings of the 10th Conference on Uncertainty in AI, Morgan Kaufmann, San Francisco, 1994, pp. 60–68.
[7] H. L. Bodlaender, *Some classes of graphs with bounded tree width*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 1988, pp. 116–126.
[8] R. B. Borie, R. G. Parker, and C. A. Tovey, *Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families*, Algorithmica, 7 (1992), pp. 555–581.
[9] A. Bhatia, *Full Degree Spanning Tree Problem*, MS paper, University of Maryland, College Park, MD, 1998.
[10] R. Bhatia, S. Khuller, R. Pless, and Y. Sussmann, *The full degree spanning tree problem*, Networks, 36 (2000), pp. 203–209.
[11] H. J. Broersma, A. Huck, T. Kloks, O. Koppius, D. Kratsch, H. Müller, and H. Tuinstra, *Degree-preserving forests*, Networks, 35 (2000), pp. 26–39.
[12] B. Chandra, H. Karloff, and C. Tovey, *New results on the old k-opt algorithm for the traveling salesman problem*, SIAM J. Comput., 28 (1999), pp. 1998–2029.
[13] F. Chudak, M. Goemans, D. Hochbaum, and D. Williamson, *Primal-dual interpretation of two* 2*-approximation algorithms for the feedback vertex set problem in undirected graphs*, Oper. Res. Lett., 22 (1998), pp. 111–118.
[14] G. Even, J. (S.) Naor, S. Rao, and B. Schieber, *Divide-and-conquer approximation algorithms via spreading metrics*, in Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1995, pp. 62–71.

[15] M. Goemans and D. Williamson, *Primal-dual approximation algorithms for feedback problems in planar graphs*, Combinatorica, 18 (1998), pp. 37–59.
[16] F. Harary, *Graph Theory*, Addison–Wesley, Reading, MA, 1969.
[17] J. Kleinberg and A. Kumar, *Wavelength conversion in optical networks*, J. Algorithms, 38 (2001), pp. 25–50.
[18] M. Lewinter, *Interpolation theorem for the number of degree-preserving vertices of spanning trees*, IEEE Trans. Circuits Systems I Fund. Theory Appl., 34 (1987), p. 205.
[19] L. E. Ormsbee, *Implicit network calibration*, J. Water Resources Planning Management, 115 (1989), pp. 243–257.
[20] L. E. Ormsbee and D. J. Wood, *Explicit pipe network calibration*, J. Water Resources Planning Management, 112 (1986), pp. 116–182.
[21] I. W. M. Pothof and J. Schut, *Graph-Theoretic Approach to Identifiability in a Water Distribution Network*, Memorandum 1283, Universiteit Twente, Twente, The Netherlands, 1995.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 488–513

LOWER BOUNDS FOR MATRIX PRODUCT IN BOUNDED DEPTH CIRCUITS WITH ARBITRARY GATES*

RAN RAZ† AND AMIR SHPILKA‡

Abstract. We prove superlinear lower bounds for the number of edges in constant depth circuits with n inputs and up to n outputs. Our lower bounds are proved for all types of constant depth circuits, e.g., constant depth arithmetic circuits and constant depth Boolean circuits with arbitrary gates. The bounds apply for several explicit functions and, most importantly, for matrix product. In particular, we obtain the following results:

1. We show that the number of edges in any constant depth arithmetic circuit for *matrix product* (over any field) is superlinear in m^2 (where $m \times m$ is the size of each matrix). That is, the lower bound is superlinear in the number of input variables. Moreover, if the circuit is bilinear, the result applies also for the case in which the circuit gets any product of two linear functions for free.
2. We show that the number of edges in any constant depth arithmetic circuit for the trace of the product of three matrices (over fields with characteristic 0) is superlinear in m^2. (Note that the trace is a *single-output* function.)
3. We give explicit examples for n Boolean functions $f_1, \ldots, f_n$, such that any constant depth *Boolean circuit with arbitrary gates* for $f_1, \ldots, f_n$ has a superlinear number of edges. The lower bound is also proved for *circuits with arbitrary gates over any finite field*. The bound applies for matrix product over finite fields as well as for several other explicit functions.

Key words. bounded depth circuits, matrix products, lower bounds, superconcentrators

AMS subject classification. 68Q17

PII. S009753970138462X

1. Introduction.

Exponential lower bounds are well known for constant depth Boolean circuits over the base $\{\text{AND}, \text{OR}, \text{NOT}\}$ [Ajt83, FSS81, Yao85, Hås86]. However, for many other types of constant depth circuits, almost nothing is known. In this work, we prove superlinear lower bounds for the number of edges in constant depth circuits with n inputs and up to n outputs. Our lower bounds are proved for all models of Boolean and arithmetic circuits and, in particular, for Boolean circuits with arbitrary gates. The bounds apply for several explicit functions and, in particular, for matrix product.

In general, our lower bound for circuits of depth $d \geq 2$ is $\Omega(n \cdot \lambda_d(n))$, where $\lambda_d(n)$ is a slowly growing function. Our main method is a graph theoretic argument that analyzes certain superconcentration properties of the circuit as a graph. Hence the same lower bounds are obtained for all types of circuits. Our results and proof methods are related to the works of [DDPW83, Pud94], where lower bounds of $\Omega(n \cdot \lambda_d(n))$ were proved for the size of superconcentrators. Pudlak used similar methods to prove lower bounds of $\Omega(n \cdot \lambda_d(n))$ for the number of edges in constant depth arithmetic circuits with n inputs and up to n outputs over fields with characteristic 0 [Pud94]. Pudlak's results hold for the parallel prefix problem as well as for other explicit functions.

In all that follows, the size of a circuit means the number of edges in it.

*Received by the editors February 6, 2001; accepted for publication (in revised form) July 1, 2002; published electronically February 20, 2003.

http://www.siam.org/journals/sicomp/32-2/38462.html

†Department of Computer Science, Weizmann Institute, Rehovot 76100, Israel (ranraz@wisdom.weizmann.ac.il). This author's research was supported by US-Israel BSF grant 98-00349 and NSF grant CCR-9987077.

‡Maxwell Dworkin 140, 33 Oxford Street, Cambridge, MA 02138 (amirs@deas.harvard.edu).

1.1. Matrix product. Matrix product is among the most studied computational problems. Surprising upper bounds of $O(m^{2+\epsilon})$ (where $\epsilon < 1$ and $m \times m$ is the size of each matrix) were obtained by Strassen in [Str69] and improved in many other works (see [Gat88] for a survey). The only known lower bound, however, is a lower bound of $2.5 \cdot m^2$ for the number of products needed [Bsh89, Bla99]. In particular, the following problem is still open: Can matrix product be computed by circuits of size $O(m^2)$? Nontrivial size-depth tradeoffs for matrix product are also unknown. In particular, the following problem is still open: Can matrix product be computed by circuits of size $O(m^2)$ and logarithmic depth? In this work, we prove that matrix product cannot be computed by circuits of size $O(m^2)$ and constant depth.

The standard computational model for matrix product is by arithmetic circuits over some field F. Usually, it is assumed that the circuits are bilinear; that is, product gates are applied only on two linear functions, where the first function is linear in the variables of the first matrix and the second function is linear in the variables of the second matrix. Such an assumption can be made without loss of generality if the field F is of characteristic 0. For fields of characteristic different than 0, the nonbilinear case is also interesting. Note, however, that all known upper bounds for matrix product (over any field) are by bilinear circuits.

In the bilinear case, our lower bound proof also works if the circuit gets any product of two linear functions for free. That is, the lower bound is proven for the number of edges above the product gates. We prove that if the circuit is of depth 1 above these products (i.e., total depth 3), it is of size $\Omega(m^3)$. For $d \geq 2$, we prove that if the circuit is of depth d above the products (i.e., total depth $d+2$), it is of size $\Omega(m^2 \cdot \lambda_d(m))$. In the general (nonbilinear) case for fields of characteristic different than 0, our lower bound is $\Omega(m^3)$ for circuits of depth 1 and $\Omega(m^2 \cdot \lambda_d(m))$ for circuits of depth $d \geq 2$. The last lower bound is a special case of a more general lower bound for circuits with arbitrary gates over finite fields. That lower bound is discussed in subsection 1.2.

1.2. Circuits with arbitrary gates. In a *Boolean circuit with arbitrary gates*, we allow each gate (of fanin k) to compute an arbitrary function $g : \{0,1\}^k \to \{0,1\}$. In this work, we give explicit examples for (up to) n Boolean functions $f_1, \ldots, f_n$, such that any constant depth Boolean circuit with arbitrary gates for $f_1, \ldots, f_n$ is of superlinear size. (As before, the bound for depth $d \geq 2$ is $\Omega(n \cdot \lambda_d(n))$.) The bound holds for matrix product over $GF(2)$ (where the dimension of each matrix is $m = \sqrt{n/2}$) as well as for matrix product over other finite fields (where, say, each field element is represented by its bits). The bound also holds for the parallel prefix problem and for other problems from [Pud94].

Previously, Impagliazzo, Paturi, and Saks [IPS97] had considered the case of circuits with arbitrary monotone gates and negations (including the case of threshold circuits). In this model, they proved superlinear lower bounds for constant depth circuits for a single Boolean function, parity. In contrast, our bounds apply to circuits with completely general gates but give only lower bounds for multiple output functions.

The above results for Boolean circuits with arbitrary gates can be generalized to circuits over larger domains. Let F be some fixed finite set (e.g., some fixed finite field), and assume that the input variables range over F (or over a subset of F). A *circuit with arbitrary gates over* F allows each gate (of fanin k) to compute an arbitrary function $g : F^k \to F$. By a reduction to the Boolean case, we get explicit examples for (up to) n functions $f_1, \ldots, f_n$, such that any constant depth circuit with

arbitrary gates over F for $f_1, \ldots, f_n$ is of superlinear size. (As before, the bound for depth $d \geq 2$ is $\Omega(n \cdot \lambda_d(n))$.) In particular, this gives lower bounds for circuits with arbitrary gates over any finite field F. The bound holds for matrix product over F as well as for many other functions.

1.3. Arithmetic circuits. In *arithmetic circuits*, the allowed gates are product and addition over a field F. Constants in the field are also allowed. Arithmetic circuits compute polynomials in the ring $F[x_1, \ldots, x_n]$ (where $x_1, \ldots, x_n$ are the input variables for the circuit), and we would like to give explicit examples for polynomials that are hard to compute. Note that for finite fields the representation of a function $f : F^n \to F$ as a polynomial is not unique (since for every i we have the equation $x_i^p = x_i$, where $p \neq 0$ is the characteristic of the field). Usually, it is required only that the circuit compute the given polynomials as functions; that is, the circuit may compute other polynomials that represent the same functions.

Lower bounds for the size of arithmetic circuits for explicit polynomials are known only if we allow polynomials with large degree or large coefficients (see, e.g., [Str73, BS82]). However, if we limit the degree and the coefficients to be of size $O(1)$, then no nontrivial lower bound is known. For constant depth arithmetic circuits, exponential lower bounds are known for fields F with characteristic $p = 2$ [Razb87, Smo87]. For other finite characteristics, exponential lower bounds are known only for depth 3 [GK98, GR98] (and, for depth larger than 3, no nontrivial lower bound was known). For characteristic 0, the best lower bounds for depth 3 are the almost quadratic bounds of $\Omega(n^{2-\epsilon})$ [SW99].

In this work, we get (for any field F) explicit examples for (up to) n polynomials $f_1, \ldots, f_n$ such that any constant depth arithmetic circuit (over F) for $f_1, \ldots, f_n$ is of superlinear size. One such example is matrix product (over F). For finite fields (and hence also for any field with characteristic different than 0), this follows by the general lower bound for circuits with arbitrary gates over F, as discussed in subsection 1.2. For fields with characteristic 0, this follows from the bilinear lower bound for matrix product, as discussed in subsection 1.1. Similar bounds for fields with characteristic 0 were previously proved by Pudlak [Pud94]. Pudlak gives explicit examples for n linear functions $f_1, \ldots, f_n$ such that any constant depth arithmetic circuit with linear gates (i.e., products are not allowed) for $f_1, \ldots, f_n$ is of superlinear size. (Over fields with characteristic 0, the assumption that all the gates in the circuit are linear can be made without loss of generality.)

For fields with characteristic 0, our results (as well as Pudlak's results) also give explicit examples for *one* polynomial $h = f_1 \cdot y_1 + \cdots + f_n \cdot y_n$ (in the input variables $x_1, \ldots, x_n, y_1, \ldots, y_n$) such that any constant depth arithmetic circuit for h is of superlinear size. This follows easily by the result of [BS82] and was noted to us by Toni Pitassi and Avi Wigderson.

1.4. Methods and related work. Our main lemma gives an analysis of the structure of a constant depth circuit as a graph. Let G be a directed acyclic graph. Denote by V_G the set of all nodes of G. Denote by I_G the set of all nodes of indegree 0 (inputs) and by O_G the set of all nodes of outdegree 0 (outputs). The depth of G is the length of the longest directed path in G. Roughly speaking, the main lemma shows that if G is of depth d and has fewer than $n \cdot \lambda_d(n)$ edges (where, for each $1 \leq d$, $\lambda_d(n)$ is a slowly growing function of n), then one can remove from G a set of $\epsilon \cdot n$ inputs and $\epsilon \cdot n$ outputs (for some small constant ϵ) and a small number of intermediate nodes, such that, in the new graph, the total number of directed paths from I_G to O_G is small.

LEMMA 1.1. *For any $0 < \epsilon < 1/400$ and any directed acyclic graph G of depth d, with more than n vertices and less than $\epsilon \cdot n \cdot \lambda_d(n)$ edges, the following is satisfied.*

For some k, such that $\sqrt{n} \leq k = o(n)$, there exist subsets $I \subset I_G$, $O \subset O_G$, and $V \subset V_G$, such that $|I|, |O| \leq 5\epsilon \cdot d \cdot n$, $|V| = k$, and the total number of directed paths from $I_G \setminus I$ to $O_G \setminus O$ that do not pass through nodes in V is at most $\epsilon \cdot n^2/k$.

Lemma 1.1 is restated (in a slightly more general form) as Corollary 3.12.

The main lemma is used to transform any circuit of depth d and size less than $\epsilon n \lambda_d(n)$ into a new circuit of depth 1 (and relatively small size). This is done by removing from the original circuit $5\epsilon dn$ inputs, $5\epsilon dn$ outputs, and a small number of intermediate nodes. The lower bounds then follow by a *rigidity argument* in the spirit of Valiant's approach [Val77].

As mentioned before, similar methods were previously used to prove lower bounds for superconcentrators [DDPW83, Pud94] and for constant depth arithmetic circuits over fields with characteristic 0 [Pud94]. In particular, methods similar to our main lemma are implicit in [Pud94] (although the presentation there is different). Versions of these methods appeared already in [DDPW83]. One can think of Lemma 1.1 also as a generalization of the lower bounds for superconcentrators given in [DDPW83, Pud94]. In fact, all these lower bounds follow easily by a reduction to Lemma 1.1. Our proof for Lemma 1.1 relies on [Pud94].

1.5. Organization of the paper. In section 2, we give the definition of the functions $\lambda_d(n)$ and prove some simple properties of these functions. In section 3, we give the proof of Lemma 1.1. In section 4, we prove our results for bilinear arithmetic circuits. In section 5, we prove our results for Boolean circuits with arbitrary gates and for circuits with arbitrary gates over finite fields.

2. Slowly growing functions. In this section, we define the functions $\lambda_d(n)$ and we prove some easy properties of them. We start with a definition of the "star" operator.

DEFINITION 2.1. *For a function f, define $f^{(i)}$ to be the composition of f with itself i times; i.e., $f^{(i)} = f \circ f \circ \cdots \circ f$ i times, $f^{(1)} = f$.*

For a function $f : N \to N$ such that $f(n) < n$ for $n > 0$, define

$$f^*(n) = \min\{\ i \mid \text{ such that } \ f^{(i)}(n) \leq 1\}.$$

We will need the following properties of f^* (taken from [Pud94]).

CLAIM 2.2 (see [Pud94]). *Suppose $f(n) \leq \lfloor \sqrt{n} \rfloor$. For every $n \geq 0$, we have*

1. $\frac{f^{(i)}(n)}{f^{(i+1)}(n)} \geq f^{(i+1)}(n)$ *for every $i > 0$ (provided that the denominator is not 0),*
2. $f^{(i)}(n) \geq \frac{f^*(n)}{2}$ *for every $i \leq \frac{f^*(n)}{2}$.*

The proof is taken from [Pud94] as well.

Proof.

1.

$$\frac{f^{(i)}(n)}{f^{(i+1)}(n)} \geq \frac{f^{(i)}(n)}{\lfloor \sqrt{f^{(i)}(n)} \rfloor} \geq \sqrt{f^{(i)}(n)} \geq f^{(i+1)}(n).$$

2. From (1) it follows that, if $f^{(i+1)}(n) > 1$, then $f^{(i)}(n) > f^{(i+1)}(n)$. Therefore,

$$f(n) > f^{(2)}(n) > \cdots > f^{(f^*(n))}(n).$$

Since the values of f are integers, the result follows. □

Our lower bounds will be expressed in terms of the following set of slowly growing functions.

DEFINITION 2.3. *Let*

$$\lambda_1(n) = \lfloor \sqrt{n} \rfloor,$$

$$\lambda_2(n) = \lceil \log n \rceil,$$

$$\lambda_d(n) = \lambda^*_{d-2}(n).$$

Some easy-to-verify properties of these functions are the following.

CLAIM 2.4.

1. *Each $\lambda_i(n)$ is a monotone increasing function tending to infinity with n.*
2. *For $i \geq 2$, $\lambda_{2i}(n) = \Theta(\lambda_{2i+1}(n))$.*
3. *For $i \geq 2$ and n large enough, $\lambda_i(n) \leq \lfloor \sqrt{\frac{n}{2}} \rfloor$.*

Proof.

1. The fact that λ_i is increasing is immediate from the fact that λ_1 and λ_2 are.
2. Notice that $\lambda_3(n) = \Theta(\log \log n)$. Since $\log \log n = \log^{(2)}(n)$, we have $\lambda_4(n) = \Theta(\lambda_5(n))$. Using induction, we get the desired result.
3. Clearly $\lambda_2(n), \lambda_3(n) \leq \sqrt{\frac{n}{2}}$ for n large enough. Assume that $\lambda_j(n) \leq \sqrt{\frac{n}{2}}$. We have

$$\sqrt{\frac{n}{2}} \geq \lambda_j(n) \geq (\lambda_j^{(2)}(n))^2 \geq \frac{1}{4}(\lambda_{j+2}(n))^2;$$

hence

$$\lambda_{j+2}(n) \leq (8n)^{\frac{1}{4}} \leq \sqrt{\frac{n}{2}}$$

for n large enough. ∎

3. Superconcentration properties of graphs. In this section, we prove our main lemma on graphs and several stronger versions of it. The lemma will be used to analyze the structure of a constant depth circuit as a graph. For simplicity, we prove here the lemma for leveled graphs. The general case follows easily by a reduction to the leveled case. Let $G = (V_G, E_G)$ be a leveled graph of depth d. The number of levels in G is hence $d+1$, and all edges in the graph are between vertices of adjacent levels. In all the following, we allow all graphs to be multigraphs.

We will use the following notation: We denote by $L_0, \ldots, L_d$ the levels of G; that is, L_i is the set of vertices at level i. The set of vertices L_0 is also denoted by I_G (and we call these vertices *inputs*). The set of vertices L_d is also denoted by O_G (and we call these vertices *outputs*).

Let $U \subset V_G$ be a set of vertices. We denote by $E(U)$ the set of edges that touch vertices in U. We denote by $\Gamma(U)$ the set of neighbors of U. We denote by $maxdeg(U)$ the maximal degree of a vertex in U. For subsets $I \subset I_G$, $O \subset O_G$, $V \subset V_G$, we denote by $P_G[I, O, V]$ the total number of paths of length d between I and O that do not pass through vertices in V.

Since our results are expressed as functions of the λ_d's, all our results hold whenever n is large enough. So, from now on, we assume that n is large enough.

3.1. Depth 2. We will prove the main lemma for graphs of depth 2. We will start by proving a stronger lemma (Lemma 3.1). The main lemma (for depth 2) will then follow as Corollary 3.3. These lemmas are not needed for the proofs for higher depth. Nevertheless, the methods used here give some hints for the proofs needed for the general case.

LEMMA 3.1. *Let $\epsilon > 0$, and let $1 \le k \le n$. Let G be a leveled graph of depth 2 with at most $\epsilon n \lambda_2(\frac{n}{k})$ edges. Assume that $|L_1| \ge k$. Then there exists a set $V \subset L_1$ of size $k \le |V| \le \sqrt{kn}$ such that*

$$P_G[I_G, O_G, V] \le \frac{100\epsilon^2 n^2}{|V|}.$$

For the proof of Lemma 3.1, we will need Lemma 3.2. For the proof of Lemma 3.2, the reader is referred to [Pud94, Lemma 4].

LEMMA 3.2 (see [Pud94]). *Let $c_1 \ge c_2 \ge \cdots \ge c_t \ge 0$ be a sequence of real numbers, and let p, q be two integers such that $1 \le p \le q \le t$. If, for every $p \le l \le q$,*

$$\sum_{i=l}^{t} {c_i}^2 \ge \frac{1}{l},$$

then

$$\sum_{i=1}^{t} c_i \ge \frac{1}{2} \log\left(\frac{q}{p}\right).$$

Proof of Lemma 3.1. Denote $m = |L_1|$. Let $v_1, v_2, \ldots, v_m$ be the vertices of L_1, ordered according to their degree, from highest to lowest. That is, for every i, $deg(v_i) \ge deg(v_{i+1})$. For every $k \le l \le \sqrt{nk}$, denote $V_l = \{v_1, \ldots, v_l\}$. Then, for every such l,

$$P_G[I_G, O_G, V_l] \le \sum_{i=l+1}^{m} (deg(v_i))^2.$$

Let $c_i = \frac{deg(v_i)}{10\epsilon n}$. Then

$$\sum_{i=1}^{m} c_i = \frac{1}{10\epsilon n} \sum_{i=1}^{m} deg(v_i) = \frac{1}{5\epsilon n}|E_G| \le \frac{1}{5}\left\lceil \log\left(\frac{n}{k}\right)\right\rceil < \frac{1}{2}\log\left(\frac{\sqrt{n}}{\sqrt{k}}\right) = \frac{1}{2}\log\left(\frac{\sqrt{nk}}{k}\right).$$

Therefore, by Lemma 3.2, for some $k \le l \le \sqrt{nk}$,

$$\sum_{i=l}^{m} {c_i}^2 < \frac{1}{l}.$$

Hence

$$P_G[I_G, O_G, V_l] \le \sum_{i=l}^{m} (deg(v_i))^2 = (100\epsilon^2 n^2) \sum_{i=l}^{m} c_i^2 \le \frac{100\epsilon^2 n^2}{l} = \frac{100\epsilon^2 n^2}{|V_l|}. \qquad \square$$

As a corollary, we obtain our main lemma for $d = 2$.

COROLLARY 3.3. *Let G be a leveled graph of depth 2 with at most $\epsilon n \lambda_2(n)$ edges for some $0 < \epsilon < 1/400$. Assume that $|L_1| \geq \sqrt{n}$. Then there exists a set $V \subset L_1$ of size $\sqrt{n} \leq |V| \leq n^{\frac{3}{4}}$ such that*

$$P_G[I_G, O_G, V] \leq \frac{\epsilon n^2}{|V|}.$$

Proof. Note that

$$\epsilon n \lambda_2(n) \leq 2\epsilon n \lambda_2\left(\frac{n}{\sqrt{n}}\right).$$

By Lemma 3.1 with $k = \sqrt{n}$, there is a set $V \subset L_1$ of size $\sqrt{n} \leq |V| \leq n^{\frac{3}{4}}$ such that

$$P_G[I_G, O_G, V] \leq \frac{100(2\epsilon)^2 n^2}{|V|} \leq \frac{\epsilon n^2}{|V|}. \qquad \square$$

3.2. Reducing the depth by 2. Roughly speaking, the main lemma shows that, if G has a small number of edges, then one can remove from G a small set J of inputs and outputs and a small set V of other vertices, such that the total number of paths of length d from $I_G \setminus J$ to $O_G \setminus J$ that do not pass through vertices in V is small (i.e., $P_G[\, I_G \setminus J, O_G \setminus J, V]$ is small).

The proof is by induction on the depth, and it uses a specific way to reduce the depth by 2: Given a partition (A, B, C) of $L_1 \cup L_{d-1}$, we will eliminate $L_1 \cup L_{d-1}$ by

1. removing A,
2. removing all neighbors of B among $I_G \cup O_G$, and
3. adding an edge between vertices from I_G and L_2 that are connected through C (and the same with L_{d-2} and O_G) and then removing C.

This leads us to the following definition.

DEFINITION 3.4. *Let G be a leveled graph of depth $d \geq 3$. Let (A, B, C) be a partition of $L_1 \cup L_{d-1}$. The graph $\hat{G}$ of depth $d-2$ is defined in the following way: The inputs of $\hat{G}$ are*

$$I_{\hat{G}} = I_G \setminus \Gamma(B).$$

The outputs of $\hat{G}$ are

$$O_{\hat{G}} = O_G \setminus \Gamma(B).$$

The $d-1$ levels of $\hat{G}$ are

$$I_{\hat{G}}, L_2, \ldots, L_{d-2}, O_{\hat{G}}$$

(note that, for $d = 3$, the levels of $\hat{G}$ are just $I_{\hat{G}}, O_{\hat{G}}$). The edges between the levels $L_2, \ldots, L_{d-2}$ are the same as they are in G. The other edges are defined in the following way:

- *For $d > 3$, we have to define the edges between $I_{\hat{G}}$ and L_2 and between L_{d-2} and $O_{\hat{G}}$. For every v, c, w such that $v \in I_{\hat{G}}$, $c \in C$, $w \in L_2$, and there are edges $(v, c), (c, w)$ in G, we add the edge (v, w) to $\hat{G}$ (i.e., we replace every such path of length 2 with an edge). In the same way, for every v, c, w such that $v \in L_{d-2}$, $c \in C$, $w \in O_{\hat{G}}$, and there are edges $(v, c), (c, w)$ in G, we add the edge (v, w) to $\hat{G}$.*

- *For $d = 3$, we have to define the edges between $I_{\hat{G}}$ and $O_{\hat{G}}$. This case is slightly different because we do not have a level between L_1 and L_2 to absorb the new edges. So instead, for every path of length 3 (v, c_1, c_2, w), such that $v \in I_{\hat{G}}$, $w \in O_{\hat{G}}$, and $c_1, c_2 \in C$, we put an edge (v, w) in $\hat{G}$.*

Clearly $\hat{G}$ is a function of G, A, B, C. For convenience, we will use the notation $\hat{G}$ instead of $\hat{G}(G, A, B, C)$.

Obviously, $\hat{G}$ is a multigraph of depth $d-2$. In the construction of $\hat{G}$, we replaced each path of length 2 through C (or a path of length 3 in the case in which $d = 3$) with an edge. Therefore, we have the following easy corollary. The corollary shows that the number of relevant paths in the graph $\hat{G}$ is the same as it is in the graph G after removing the sets A and $\Gamma(B)$. Hence, in order to count the paths in G, it is enough to count the corresponding paths in $\hat{G}$. This easy observation makes the use of induction possible.

PROPOSITION 3.5. *Let $G, A, B, C, \hat{G}$ be as in Definition* 3.4. *Then the following hold:*

1. $P_G[\, I_G \setminus \Gamma(B),\, O_G \setminus \Gamma(B),\, A] = P_{\hat{G}}[I_{\hat{G}}, O_{\hat{G}}, \emptyset]$.
2. *More generally, for any set of vertices $\hat{V} \subset L_2 \cup \cdots \cup L_{d-2}$ and any set of inputs and outputs $\hat{J} \subset I_{\hat{G}} \cup O_{\hat{G}}$,*

$$P_G[\, I_G \setminus (\hat{J} \cup \Gamma(B)),\, O_G \setminus (\hat{J} \cup \Gamma(B)),\, \hat{V} \cup A] = P_{\hat{G}}[\, I_{\hat{G}} \setminus \hat{J},\, O_{\hat{G}} \setminus \hat{J},\, \hat{V}].$$

As mentioned above, we will prove the main lemma by induction. In each step, we reduce the depth by 2 by removing a set A of intermediate vertices and a set $\Gamma(B)$ of inputs and outputs. The final set V will be the union of the sets A from all steps of the induction, and the final set J will be the union of the sets $\Gamma(B)$ from all these steps.

In each step of the induction, we assume that the graph G has a relatively small number of edges. We would like to make sure that $\hat{G}$ also has a small number of edges. The next lemma shows that, given certain bounds for $|A|$, $maxdeg(C)$, and $|E_G|$, one can bound the number of edges in $\hat{G}$. The idea of the proof is that the only edges that we add to $\hat{G}$ are related to paths through C. Therefore (roughly), we can bound $|E_{\hat{G}}|$ by $|E_G| \cdot maxdeg(C)$. Note that we assume here the bound

$$|E_G| \leq \epsilon n \lambda_d \left(\frac{n}{k}\right)$$

for some $k \geq 1$. This is more general than the original assumption $|E_G| \leq \epsilon n \lambda_d(n)$. The reason that we need a more general assumption is that the graph G may be a graph that was obtained after several steps of reduction (rather than the original graph). Roughly, the parameter k corresponds to the number of intermediate vertices that were already removed from the original graph (the set V in the statement of the lemma). Since we think of the set A as being removed from the graph as well (and being added to the set V), the bound that we want for $|E_{\hat{G}}|$ is

$$|E_{\hat{G}}| \leq \epsilon n \lambda_{d-2} \left(\frac{n}{k + |A|}\right).$$

LEMMA 3.6. *Let $G, A, B, C, \hat{G}$ be as in Definition* 3.4. *Let $0 < \epsilon < 1/3$ and $1 \leq k \leq \epsilon^2 n$. Assume that*

$$|E_G| \leq \epsilon n \lambda_d \left(\frac{n}{k}\right)$$

and that, for some integer $1 \leq i \leq \frac{\lambda_d(\frac{n}{k})}{2} - 3$*, we have*

1. $|A| \leq \frac{2\epsilon n}{\lambda^{(i)}_{d-2}(\frac{n}{k})}$ *and*
2. $maxdeg(C) \leq \lambda^{(i+3)}_{d-2}(\frac{n}{k})$.

Then

$$|E_{\hat{G}}| \leq \epsilon n \lambda_{d-2}\left(\frac{n}{k+|A|}\right).$$

The proof is by a straightforward calculation using Claims 2.2 and 2.4.

Proof. Since $k \leq \epsilon^2 n$, we have

$$k\lambda^{(i)}_{d-2}\left(\frac{n}{k}\right) \leq k\lambda_1\left(\frac{n}{k}\right) \leq k\sqrt{\frac{n}{k}} < \epsilon n;$$

therefore,

$$k + |A| \leq k + \frac{2\epsilon n}{\lambda^{(i)}_{d-2}(\frac{n}{k})} = \frac{k\lambda^{(i)}_{d-2}(\frac{n}{k}) + 2\epsilon n}{\lambda^{(i)}_{d-2}(\frac{n}{k})} < \frac{3\epsilon n}{\lambda^{(i)}_{d-2}(\frac{n}{k})} < \frac{n}{\lambda^{(i)}_{d-2}(\frac{n}{k})}.$$

We now have two different cases that follow from the construction in Definition 3.4:

1. $d > 3$. From the construction of $\hat{G}$, we get that the degree of each vertex in $L_2 \cup L_{d-2}$ has increased by a factor of $maxdeg(C)$ at most. Therefore, we have

$$|E_{\hat{G}}| \leq |E_G| \cdot maxdeg(C) \leq \epsilon n \lambda_d\left(\frac{n}{k}\right) \lambda^{(i+3)}_{d-2}\left(\frac{n}{k}\right)$$

$$\leq 2\epsilon n \left(\lambda^{(i+3)}_{d-2}\left(\frac{n}{k}\right)\right)^2 \leq \epsilon n \lambda^{(i+2)}_{d-2}\left(\frac{n}{k}\right) \leq \epsilon n \lambda^{(2)}_{d-2}\left(\frac{n}{k+|A|}\right)$$

$$< \epsilon n \lambda_{d-2}\left(\frac{n}{k+|A|}\right)$$

(where all inequalities are due to Claims 2.2 and 2.4 and the bound that we proved on $k + |A|$).

2. $d = 3$. Since we dropped $L_1 \cup L_2$, the set of inputs and outputs now absorbs both the edges of $C \cap L_1$ and the edges of $C \cap L_2$, so we have

$$|E_{\hat{G}}| \leq |E_G| \cdot maxdeg(C)^2 \leq \epsilon n \lambda_d\left(\frac{n}{k}\right)\left(\lambda^{(i+3)}_{d-2}\left(\frac{n}{k}\right)\right)^2$$

$$\leq \frac{1}{2}\epsilon n \lambda_d\left(\frac{n}{k}\right) \lambda^{(i+2)}_{d-2}\left(\frac{n}{k}\right) \leq \frac{1}{2}\epsilon n \lambda^{(i+1)}_{d-2}\left(\frac{n}{k}\right)$$

$$\leq \frac{1}{2}\epsilon n \lambda_{d-2}\left(\frac{n}{k+|A|}\right)$$

(where, as before, all inequalities are due to Claims 2.2 and 2.4 and the bound that we proved on $k + |A|$). ∎

So far, we have presented the construction of $\hat{G}$ from G, given an arbitrary partition (A, B, C). In order to maintain the bound on the number of edges of $\hat{G}$, we need A, C to satisfy the conditions of Lemma 3.6. Also, we need $\Gamma(B)$ to be not too large in order to make sure that the total number of inputs and outputs removed in the process is small. The next lemma shows how to partition $L_1 \cup L_{d-1}$ into suitable sets (A, B, C). The way it is done is by first ordering the vertices in $L_1 \cup L_{d-1}$ according to their degrees (from highest to lowest) and then finding appropriate numbers $r_1 > r_2$, such that A will be the set of vertices with degree larger than r_1, B will be the set of vertices with degree larger than r_2 and at most r_1, and C will be the set of vertices with degree at most r_2.

LEMMA 3.7. *Let G be a leveled graph of depth $d \geq 3$, such that $|E_G| \leq \epsilon n \lambda_d(r)$ for large enough r (more accurately, we need $\lambda_d(r) > 72$). Then there exist a partition (A, B, C) of $L_1 \cup L_{d-1}$ and $1 \leq i \leq \lambda_d(r)/2 - 3$ with the following properties:*

1. $|A| \leq \frac{2\epsilon n}{\lambda_{d-2}^{(i)}(r)}$,
2. $|\Gamma(B)| \leq 9\epsilon n$,
3. $maxdeg(C) \leq \lambda_{d-2}^{(i+3)}(r)$.

Proof. Denote

$$W_0 = \{v \in L_1 \cup L_{d-1} \mid \ deg(v) > \lambda_{d-2}(r)\}$$

and, for $i \geq 1$,

$$W_i = \{v \in L_1 \cup L_{d-1} \mid \ \lambda_{d-2}^{(i)}(r) \geq deg(v) > \lambda_{d-2}^{(i+1)}(r)\}.$$

CLAIM 3.8. *For every $1 \leq i \leq \lambda_d(r)/2 - 3$,*

$$|W_0 \cup W_1 \cup \cdots \cup W_{i-1}| \leq \frac{2\epsilon n}{\lambda_{d-2}^{(i+1)}(r)}.$$

Proof. The proof follows from the fact that the degree of each vertex in $W_0 \cup W_1 \cup \cdots \cup W_{i-1}$ is at least $\lambda_{d-2}^{(i)}(r)$. If the claim were not true, we would have had

$$|E_G| \geq |E(W_0 \cup W_1 \cup \cdots \cup W_{i-1})|\lambda_{d-2}^{(i)}(r) \geq \frac{2\epsilon n}{\lambda_{d-2}^{(i+1)}(r)}\lambda_{d-2}^{(i)}(r)$$

$$\geq 2\epsilon n \lambda_{d-2}^{(i+1)}(r) \geq \epsilon n \lambda_d(r),$$

in contradiction. ∎

CLAIM 3.9. *For some $0 \leq i \leq \lambda_d(r)/2 - 4$,*

$$|E(W_i \cup W_{i+1} \cup W_{i+2} \cup W_{i+3})| \leq 9\epsilon n.$$

Proof. The proof follows from the bound on $|E_G|$. Since $|E_G| \leq \epsilon n \lambda_d(r)$, we must have

$$\sum_{i=0}^{\frac{\lambda_d(r)/2-4}{4}} |E(W_{4i} \cup W_{4i+1} \cup W_{4i+2} \cup W_{4i+3})| \leq \epsilon n \lambda_d(r).$$

If each of the sets $E(W_{4i} \cup W_{4i+1} \cup W_{4i+2} \cup W_{4i+3})$ were of size larger than $9\epsilon n$, then we would have had

$$|E_G| \geq 9\epsilon n \frac{\lambda_d(r)/2 - 4}{4} > \epsilon n \lambda_d(r)$$

(for large enough r), in contradiction. ∎

Fix i' to be such that Claim 3.9 is satisfied for i'. The proof of the lemma now follows for $i = i' + 1$ by taking

1. $A = W_0 \cup W_1 \cup \cdots \cup W_{i'-1}$,
2. $B = W_{i'} \cup W_{i'+1} \cup W_{i'+2} \cup W_{i'+3}$,
3. $C = (L_1 \cup L_{d-1}) \setminus (A \cup B)$. ∎

3.3. Depth 3. The induction that we are about to perform on the depth will end with a graph of depth 2 or 3 (without loss of generality, 3). We will now give the proof of the main lemma for the special case in which $d = 3$. As mentioned before, we need a more general lemma that assumes a more general bound for the number of edges in the graph.

The proof for $d = 3$ already gives the main idea of the proof for the general case. First, we partition $L_1 \cup L_2$ into three sets (A, B, C) as described before. The partition (A, B, C) will satisfy the following: The first set A is small, the second set B is not connected to many inputs and outputs, and each vertex in the third set C has small degree. We then reduce the depth of the graph to 1, using Definition 3.4. Thus we get a graph, $\hat{G}$, with the same number of input-output paths, between $I_{\hat{G}}$ and $O_{\hat{G}}$. Since $\hat{G}$ is a graph of depth 1, the number of such paths is simply the number of edges. The last step would be to calculate the number of edges in $\hat{G}$, which is done using Lemma 3.6.

LEMMA 3.10. *Let* $0 < \epsilon < 1/3$. *Let* $0 < \beta$ *and* $0 < \alpha$ *satisfy*

$$\alpha < \epsilon^2 \text{ and } \frac{4\epsilon}{\lambda_3(\frac{1}{\alpha})} < \beta.$$

Let G *be a leveled graph of depth* 3 *with at most* $\epsilon n \lambda_3(\frac{n}{k})$ *edges for some* $1 \le k \le \alpha n$. *Then there exists a partition* (A, B, C) *of* $L_1 \cup L_2$ *such that*

1. $|A| < \beta n$,
2. $|\Gamma(B)| \le 9\epsilon n$,
3. $P_G[\, I_G \setminus \Gamma(B),\, O_G \setminus \Gamma(B),\, A] \le \epsilon n \lambda_1(\frac{n}{k+|A|})$.

Proof. By Lemma 3.7, we can partition $L_1 \cup L_2$ into three sets (A, B, C) such that, for some $1 \le i \le \lambda_3(\frac{n}{k})/2 - 3$, we have

1. $|A| \le \frac{2\epsilon n}{\lambda_1^{(i)}(\frac{n}{k})} \le \frac{2\epsilon n}{\lambda_1^{(i)}(\frac{1}{\alpha})} \le \frac{4\epsilon n}{\lambda_3(\frac{1}{\alpha})} < \beta n$,
2. $|\Gamma(B)| \le 9\epsilon n$,
3. $maxdeg(C) \le \lambda_1^{(i+3)}(\frac{n}{k})$.

Let $\hat{G}$ be as in Definition 3.4 (with respect to (A, B, C)). By Proposition 3.5,

$$P_G[\, I_G \setminus \Gamma(B),\, O_G \setminus \Gamma(B),\, A] = P_{\hat{G}}[I_{\hat{G}}, O_{\hat{G}}, \emptyset].$$

Since $\hat{G}$ is of depth 1, the right-hand side equals $|E_{\hat{G}}|$. By Lemma 3.6,

$$|E_{\hat{G}}| \le \epsilon n \lambda_1 \left(\frac{n}{k + |A|} \right).$$

Hence

$$P_G[\, I_G \setminus \Gamma(B),\, O_G \setminus \Gamma(B),\, A] \le \epsilon n \lambda_1 \left(\frac{n}{k + |A|} \right). \quad ∎$$

3.4. Higher depth. We are now ready to state and prove our main lemma. As mentioned above, we actually prove a stronger lemma that will be needed for the induction. The main lemma will then follow as an easy corollary. First, note that the functions $\lambda_d(n)$ satisfy $\lambda_{2i}(n) = \Theta(\lambda_{2i+1}(n))$ for $2 \leq i$. Hence we can assume without loss of generality that the depth d is odd; otherwise, we can just increase the depth by 1. (We could not prove better results for the even levels because of the constructions given in [DDPW83].)

LEMMA 3.11. *Let $0 < \epsilon < 1/3$. For any odd integer $3 \leq d$ and $0 < \beta < 1$, there exists $0 < \alpha = \alpha(d, \beta)$ such that, if $1 \leq k \leq \alpha n$ and G is a leveled graph of depth d, with at most $\epsilon n \lambda_d(\frac{n}{k})$ edges, then there exist a set V of vertices and a set J of inputs and outputs such that*

1. $|V| \leq \beta n$,
2. $|J| \leq 5\epsilon d n$,
3. $P_G[\, I_G \setminus J,\, O_G \setminus J,\, V] \leq \epsilon n \lambda_1(\frac{n}{k+|V|})$.

Proof. The proof is by induction on d. The base case $d = 3$ was proved in Lemma 3.10. So assume that, for any $0 < \beta < 1$ and any integer $0 < l$, $\alpha(d - 2l, \beta)$ exists. Let α satisfy the following:

$$\alpha + \frac{4\epsilon}{\lambda_d(\frac{1}{\alpha})} < \alpha\left(d - 2, \frac{\beta}{2}\right) \text{ and}$$

$$\frac{\beta}{2} + \frac{4\epsilon}{\lambda_d(\frac{1}{\alpha})} < \beta.$$

Clearly such an α exists. Take $\alpha(d, \beta) = \alpha$. Let G be a leveled graph of depth d, with at most $\epsilon n \lambda_d(\frac{n}{k})$ edges, for $1 \leq k \leq \alpha n$. Let (A, B, C) be the partition of $L_1 \cup L_{d-1}$ from Lemma 3.7. Let $\hat{G}$ be the depth $d-2$ graph defined in Definition 3.4 (with respect to (A, B, C)). Then, by Lemma 3.6,

$$|E_{\hat{G}}| \leq \epsilon n \lambda_{d-2}\left(\frac{n}{k + |A|}\right).$$

Also, for some

$$1 \leq i \leq \frac{1}{2}\lambda_{d-2}\left(\frac{n}{k}\right) - 3,$$

we have that

$$|A| \leq \frac{2\epsilon n}{\lambda^{(i)}_{d-2}(\frac{n}{k})} \leq \frac{2\epsilon n}{\lambda^{(i)}_{d-2}(\frac{1}{\alpha})} \leq \frac{4\epsilon n}{\lambda_d(\frac{1}{\alpha})}.$$

Therefore,

$$k + |A| \leq \alpha n + \frac{4\epsilon n}{\lambda_d(\frac{1}{\alpha})} \leq \alpha\left(d - 2, \frac{\beta}{2}\right) \cdot n.$$

Hence the inductive assumption holds for $\hat{G}$ with parameters $d - 2$ and $\frac{\beta}{2}$. We get that, for $\hat{G}$, there exist a set $\hat{V}$ of vertices and a set $\hat{J}$ of inputs and outputs such that

1. $|\hat{V}| \leq \frac{\beta}{2} n$,

2. $|\hat{J}| \leq 5\epsilon(d-2)n$,
3. $P_{\hat{G}}[\, I_{\hat{G}} \setminus \hat{J},\, O_{\hat{G}} \setminus \hat{J},\, \hat{V}] \leq \epsilon n \lambda_1(\frac{n}{k+|A|+|\hat{V}|})$.

Define $V = \hat{V} \cup A$ and $J = \hat{J} \cup \Gamma(B)$. We have that

$$|V| = |\hat{V}| + |A| \leq \frac{\beta}{2} n + \frac{4\epsilon n}{\lambda_d(\frac{1}{\alpha})} \leq \beta n.$$

Since $|\hat{J}| \leq 5\epsilon(d-2)n$ and $|\Gamma(B)| \leq 9\epsilon n$, we have that $|J| \leq 5\epsilon d n$.

Finally, by Proposition 3.5,

$$P_G[\, I_G \setminus J,\, O_G \setminus J,\, V] = P_{\hat{G}}[\, I_{\hat{G}} \setminus \hat{J},\, O_{\hat{G}} \setminus \hat{J},\, \hat{V}]$$

$$\leq \epsilon n \lambda_1 \left(\frac{n}{k + |A| + |\hat{V}|} \right) = \epsilon n \lambda_1 \left(\frac{n}{k + |V|} \right). \qquad \square$$

Our main lemma is now stated as the following corollary. Note that the requirement $\epsilon < 1/400$ is needed only for the case in which $d = 2$. (A weaker requirement is needed for $d > 2$.)

COROLLARY 3.12. *Let $0 < \beta < 1$. Let G be a leveled graph of constant depth $d \geq 2$, with more than n vertices and less than $\epsilon n \lambda_d(n)$ edges, for some $0 < \epsilon < 1/400$ and n sufficiently large. Then there exist a set V of vertices and a set J of inputs and outputs such that*

1. $\sqrt{n} \leq |V| \leq \beta n$,
2. $|J| \leq 5\epsilon d n$,
3. $P_G[\, I_G \setminus J,\, O_G \setminus J,\, V] \leq \epsilon \frac{n^2}{|V|}$.

Proof. First note that (as mentioned above) Lemma 3.11 is correct also for even depth if we just require ϵ to be slightly smaller (a factor of 2 or 3 is enough). Since here we require $\epsilon < 1/400$, we can apply Lemma 3.11 for any depth larger than 2.

We apply Lemma 3.11 with $k = 1$. If $|V| \geq \sqrt{n}$, we are done. Otherwise, just add arbitrary vertices to V. Call the resulting set V'. We have

$$P_G[\, I_G \setminus J,\, O_G \setminus J,\, V'] \leq P_G[\, I_G \setminus J,\, O_G \setminus J,\, V]$$

$$\leq \epsilon n \lambda_1 \left(\frac{n}{1 + |V|} \right) \leq \epsilon n \sqrt{\frac{n}{1 + |V|}} = \epsilon \frac{n^2}{\sqrt{n(1 + |V|)}} \leq \epsilon \frac{n^2}{|V'|}.$$

This completes the proof for depth higher than 2. For $d = 2$, the corollary was already stated as Corollary 3.3. $\square$

3.5. Graphs for matrix product. As mentioned in the introduction, the main function that we concentrate on in this work is matrix product. A circuit for matrix product has $2m^2$ inputs and m^2 outputs, where m is the dimension of each matrix. Therefore, we will assume here that our graph has $2m^2$ inputs and m^2 outputs and that the outputs are ordered as a matrix. We will refer to such a graph as a graph for matrix product. For convenience, we will prove a lemma that will be specific for such graphs. The proof will follow easily by Corollary 3.12.

Denote by O_i the outputs in the ith column of the output matrix. Denote by $[m]$ the set $\{1, \ldots, m\}$. We think of $[m]$ as the set of all output columns. For a subset $D \subset [m]$, denote by O_D the outputs in all the columns in D. That is, $O_D = \cup_{i \in D} O_i$.

Roughly, our lemma will state that, after removing from the graph a small set I of inputs, a small set O of outputs, and a set V of size k of intermediate nodes, one can find a set D of $10k/m$ output columns such that there are no paths between the inputs and the outputs in O_D. For simplicity, we will not state the lemma for a general constant ϵ, and we will just fix some constant that will be good enough.

LEMMA 3.13. *Let G be a leveled graph for matrix product, of constant depth $d \geq 2$, with less than $\epsilon m^2 \lambda_d(m^2)$ edges, for $\epsilon = 1/(1000 \cdot d)$. Then, for any $0 < \beta < 1$, there exist sets $V \subset V_G$, $D \subset [m]$, $O \subset O_D$, $I \subset I_G$ such that the following hold:*

1. $m \leq |V| \leq \beta m^2$.
2. $|D| \geq \frac{10|V|}{m}$.
3. *For every $i \in D$, $|O_i \cap O| < \frac{1}{10}m$.*
4. $|I| < \frac{1}{10}m^2$.
5. $P_G[\, I_G \setminus I,\, O_D \setminus O,\, V] = 0$.

Proof. Let V, J be the sets guaranteed by Corollary 3.12 with $n = m^2$. Define $\tilde{I} = J \cap I_G$ and $\tilde{O} = J \cap O_G$, and let $k = |V|$. Then

1. $m \leq |V| = k \leq \beta n$ (hence requirement 1 is satisfied),
2. $|\tilde{I}|, |\tilde{O}| \leq (1/200) \cdot m^2$,
3. $P_G[\, I_G \setminus \tilde{I},\, O_G \setminus \tilde{O},\, V] \leq \epsilon \frac{m^4}{k}$.

Denote by $\tilde{D}$ the set of all $i \in [m]$ such that $|O_i \cap \tilde{O}| < (1/100) \cdot m$. Then, by the bound we have on $|\tilde{O}|$, we know that

$$|\tilde{D}| \geq (1/2) \cdot m.$$

For every $i \in \tilde{D}$, let $P(i)$ be the total number of paths between outputs in $O_i \setminus \tilde{O}$ and inputs in $I_G \setminus \tilde{I}$ that do not pass through V. Denote by D the set of $\lceil 10k/m \rceil$ indices i with the smallest $P(i)$. (Then, by the definition of D, requirement 2 is satisfied.) Denote $O = \tilde{O} \cap O_D$. (Then, by the fact that $D \subset \tilde{D}$, requirement 3 is satisfied.) Since

$$\sum_{i \in \tilde{D}} P(i) \leq P_G[\, I_G \setminus \tilde{I},\, O_G \setminus \tilde{O},\, V] \leq \epsilon \frac{m^4}{k},$$

we have

$$\sum_{i \in D} P(i) \leq \frac{|D|}{|\tilde{D}|} \sum_{i \in \tilde{D}} P(i) \leq \frac{\lceil 10k/m \rceil}{m/2} \cdot \epsilon \frac{m^4}{k} < (1/50) \cdot m^2.$$

Hence, if we denote by $\hat{I}$ the set of all inputs that are connected by a path (that do not pass through V) to some output in $O_D \setminus O$, we have

$$|\hat{I}| < (1/50) \cdot m^2.$$

Denote $I = \tilde{I} \cup \hat{I}$. (Then, by the bounds we have on $|\tilde{I}|, |\hat{I}|$, requirement 4 is satisfied.) By the definition of $\hat{I}$, all paths between $O_D \setminus O$ and $I_G \setminus I$ pass through V. (Hence requirement 5 is satisfied.) ∎

4. Arithmetic model. In this section, we present our results for bilinear arithmetic circuits. An arithmetic circuit is a directed acyclic graph as follows. Nodes of indegree 0 are called inputs and are labeled with input variables. Nodes of outdegree 0 are called outputs. Each edge is labeled with a constant from the field, and each node other than an input is labeled with one of the following operations: $\{+, \cdot\}$. (In the

first case the node is a plus gate and in the second case a product gate.) The computation is done in the following way: An input just computes the value of the variable that labels it. Then, if $v_1, \ldots, v_k$ are the vertices that fan into v, then we multiply the result of each v_i by the value of the edge that connects it to v. If v is a plus gate, we sum all the results; otherwise, v is a product gate, and we multiply all the results. Obviously, each node in the circuit computes a polynomial in the input variables.

In this section, we prove lower bounds on the size of circuits computing the product of two $m \times m$ matrices. The input is of size $n = 2m^2$, and it consists of two $m \times m$ matrices X, Y. The output is the matrix $Z = X \cdot Y$; i.e., there are m^2 outputs, and the (i, j)th output is

$$z_{i,j} = \sum_{k=1}^{m} x_{i,k} \cdot y_{k,j}.$$

Each output $z_{i,j}$ is hence a bilinear form in X and Y.

Since the product of two matrices is a bilinear form, it is natural to consider bilinear arithmetic circuits for it. A bilinear arithmetic circuit is an arithmetic circuit with the additional restriction that a product gate is allowed only to compute the product of two linear functions—one in the variables of X and the other in the variables of Y. Thus bilinear circuits have the following structure: First, there are many plus gates computing linear forms in X and linear forms in Y. Then there is one level of product gates which compute bilinear forms, and, finally, there are many plus gates that eventually compute the outputs. We will now define the *size* and *depth* of the circuit.

DEFINITION 4.1. *For a bilinear circuit C, we denote by $s(C)$ (the* size *of C) the number of edges between the product gates and the outputs. We denote by $d(C)$ (the* depth *of C) the length of the longest directed path from a product gate to an output.*

Note that these definitions ignore all gates and edges below the product gates (i.e., between the inputs and the products). That is, we allow the circuit to get for free any number of linear functions in the variables of X, and any number of linear functions in the variables of Y. We count only the size and depth above the product gates.

The requirement that the circuit is bilinear seems restrictive. It is easy to show, however, that over fields of characteristic 0, the bilinearity assumption does not change (up to a constant factor) the size and the depth of the circuit. More accurately, by paying a constant factor in the size and in the depth, we can transform any arithmetic circuit computing a bilinear form into an equivalent bilinear circuit. Roughly, this is done in the following way: Since the circuit computes polynomials of degree two, it does not really need to keep track of any monomial of higher degree. Therefore, we need only keep track of monomials of degree one or two, and we can do that by replacing each gate by a constant number of new gates (at most 5 new gates) that satisfy the bilinearity assumption. Thus we have the following proposition.

PROPOSITION 4.2. *If a set of bilinear forms is computed by an arithmetic circuit of depth d and size s over a field of characteristic 0, then there is a bilinear circuit of depth $3d$ and size $5s$ over the same field computing the same set of bilinear forms.*

Over finite fields, the bilinearity assumption may be restrictive. We prove lower bounds for the general case of arithmetic circuits over finite fields in section 5. It is also worth noting that all known algorithms for matrix product are by bilinear arithmetic circuits.

Our main tool in proving our lower bounds is Lemma 3.13. Since that lemma is stated for leveled graphs, we would like our circuit to be leveled. We hence assume that our circuit is a leveled bilinear arithmetic circuit. Since we consider leveled circuits, the depth of the circuit is just the number of levels above the product gates. Our main result in this section is the following lower bound.

THEOREM 4.3. *Any leveled bilinear arithmetic circuit C of depth d, for the product of two $m \times m$ matrices, is of size*

$$s(C) = \begin{cases} \Omega(m^3), & d = 1, \\ \Omega(\frac{1}{d} m^2 \lambda_d(m^2)), & d > 1. \end{cases}$$

In order to prove lower bounds for a nonleveled bilinear circuit, we just level it. We can do that by increasing its size by a factor of d. We can then use the lower bounds for leveled circuits. We hence have the following corollary.

COROLLARY 4.4. *Any bilinear arithmetic circuit C of depth d, for the product of two $m \times m$ matrices, is of size*

$$s(C) = \begin{cases} \Omega(m^3), & d = 1, \\ \Omega(\frac{1}{d^2} m^2 \lambda_d(m^2)), & d > 1. \end{cases}$$

In particular, this gives the following size-depth tradeoff: there is no linear size and constant depth bilinear arithmetic circuit for the product of two matrices. Note that the theorem is valid for any field. It just needs the bilinearity assumption.

After proving the main theorem, we will use the result of Baur and Strassen [BS82] to prove a lower bound on the size of bounded depth arithmetic circuits for the trace of the product of three matrices:

$$\sum_{i,j,k=1}^{m} x_{i,j} \cdot y_{j,k} \cdot z_{k,i},$$

which is a function with a single output.

Let us start with bilinear circuits of depth 1. The structure of such circuits is very simple. First, they compute linear forms. Then there is one level of product gates computing bilinear forms. Finally, there is one level of m^2 plus gates computing the outputs. For the proof, we will use the following notation:

$$O_j = \{z_{i,j} \mid i \in \{1, \ldots, m\}\};$$

i.e., O_j denotes the set of outputs of the jth column of the output matrix.

THEOREM 4.5. *Any leveled bilinear circuit C of depth 1, for the product of two $m \times m$ matrices, is of size*

$$s(C) = \Omega(m^3).$$

Proof. Since the circuit is of depth 1, the outputs come right after the product gates. Each output is computed by a plus gate that adds the results of the product gates that are connected to it. We will show that there are at least m^2 edges connected to each output column O_j. Hence, since there are m output columns, the result follows.

Assume, for a contradiction, that an output column O_j is connected to $r < m^2$ product gates. Denote the functions computed by these gates by $M_1, \ldots, M_r$. For

each M_k, denote by $L_{k,1}(X), L_{k,2}(Y)$ the two linear functions that it multiplies. That is,

$$M_k(X,Y) = L_{k,1}(X) \cdot L_{k,2}(Y).$$

Since $r < m^2$, we can find a substitution for the matrix X such that the following hold:

1. $X \neq 0$.
2. For every $1 \leq k \leq r$, $L_{k,1}(X) = 0$.

Hence, for every $1 \leq k \leq r$,

$$M_k(X,Y) = 0.$$

Therefore, $O_j = 0$, no matter what Y is. On the other hand, $X \neq 0$, and hence we can find a substitution for the matrix Y such that no column of $X \cdot Y$ is all zero (a contradiction). □

The main idea of the proof for depth 1 was the following: if there is a small number of edges in the circuit, then one can find a substitution for the matrix X such that a large number of outputs are forced to be 0 (no matter what Y is). The main idea of the proof for larger depth is the following: First, apply Lemma 3.13 to transform the circuit into a circuit of depth 1. This is done by removing from the circuit a certain number of inputs, outputs, and intermediate gates. Then use the argument for depth 1. However, since we remove from the circuit a certain number of nodes, we will need a more general argument. Roughly, we will need to generalize the proof for depth 1 to the case in which X and Y are restricted to certain subspaces of matrices. We will show that even if X and Y are restricted to (not too small) subspaces, their product still cannot be computed by a small circuit of depth 1. We will use the following notation.

DEFINITION 4.6. *For a matrix X, denote by $(X)_j$ the jth column of X. For a linear subspace of matrices $\mathcal{A}$, denote*

$$(\mathcal{A})_j = \{(X)_j \mid \ X \in \mathcal{A}\}.$$

Since $\mathcal{A}$ is a linear subspace, so is $(\mathcal{A})_j$.

We clearly have the following proposition.

PROPOSITION 4.7. *For any linear subspace of matrices $\mathcal{A}$,*

$$\dim(\mathcal{A}) \leq \sum_{j=1}^{m} \dim((\mathcal{A})_j).$$

We will also need the following lemma. Roughly, the lemma claims that if X is a matrix of large rank and $\mathcal{B}$ is a subspace of matrices of high dimension, then for many columns j, $\dim(\{(X \cdot Y)_j \mid Y \in \mathcal{B}\})$ is high.

LEMMA 4.8. *Let X be an $m \times m$ matrix of rank $\geq \frac{2}{3}m$. Let $\mathcal{B}$ be a linear subspace of $m \times m$ matrices such that $\dim(\mathcal{B}) \geq m^2 - k$. Then, for any subset of columns D of size $|D| \geq \frac{10k}{m}$, there exists a column $j \in D$ such that*

$$\dim(\{(X \cdot Y)_j \mid \ Y \in \mathcal{B}\}) \geq \frac{m}{2}.$$

Proof. Let D be a subset of columns such that $|D| \geq \frac{10k}{m}$. We first show that there is a column $j \in D$ such that $\dim((\mathcal{B})_j) \geq \frac{9}{10}m$. If this were not the case, then

by Proposition 4.7 we would have had

$$m^2 - k \leq \dim(\mathcal{B}) \leq |D| \left(\frac{9}{10} m - 1 \right) + (m - |D|)m = m^2 - \frac{m}{10}|D| - |D| < m^2 - k.$$

Let $j \in D$ be such that $\dim((\mathcal{B})_j) \geq \frac{9}{10}m$. Since $\operatorname{rank}(X) \geq \frac{2}{3}m$, we have

$$\dim(\{(X \cdot Y)_j \mid \ Y \in \mathcal{B}\}) = \dim(\{X \cdot v \mid \ v \in (\mathcal{B})_j\}) \geq \operatorname{rank}(X) - (m - \dim((\mathcal{B})_j))$$

$$\geq \frac{2}{3}m - \frac{1}{10}m > \frac{m}{2}. \qquad \square$$

In the proof for depth 1, we had $X \neq 0$. Since, in the proof for higher depth, Y will be restricted to a subspace of matrices, we will need X to satisfy a stronger condition. Namely, we need X to be of high rank. However, X itself will also be restricted to a subspace of matrices. Therefore, we want to show that in any subspace (of matrices) of high dimension, there is a matrix of high rank.

LEMMA 4.9. *In any subspace of $m \times m$ matrices of dimension larger than $(2mr - r^2 + m)$, there is a matrix of rank at least r.*

Proof. We have two different proofs. The first is for finite fields, and the second is for fields of characteristic 0. We wish to compare the number of matrices with rank at most r to the number of matrices in our linear subspace. If we prove that the number of matrices in the linear subspace is larger, then it must contain a matrix of rank larger than r.

- Assume that F is a finite field. Denote $|F| = q$. The number of elements in a subspace of dimension larger than $(2mr - r^2 + m)$ is larger than $q^{2mr - r^2 + m}$. We will now count the number of $m \times m$ matrices with rank at most r. Note that, for every such matrix, there are r rows such that every row in the matrix is in their linear span. There are $\binom{m}{r}$ possible ways to choose these r rows. There are q^{mr} possible ways to choose the r vectors for these rows. Every other row is in the linear span of these r rows, so it can be one of q^r vectors. Therefore, the number of matrices of rank at most r is bounded from above by

$$\binom{m}{r} q^{mr} (q^r)^{m-r} < q^{2mr - r^2 + m}.$$

- Assume that F is a field of characteristic 0. Instead of counting, we will consider the dimension of the variety of matrices with rank at most r. The same argument as above shows that this variety is included in the union of $\binom{m}{r}$ varieties, each of dimension at most $mr + r(m - r)$. (As before, mr is for the freedom of choice of the first r vectors, and $r(m - r)$ is for spanning the other $m - r$ rows.) Therefore, the dimension is $2mr - r^2 < 2mr - r^2 + m$. $\square$

Before giving the formal proof for Theorem 4.3, let us first give a sketch of this proof. Let C be a leveled bilinear arithmetic circuit of depth d for the product of two $m \times m$ matrices. Let G be the leveled graph of depth d, corresponding to the graph of the circuit between the product gates and the outputs (i.e., the product gates are the inputs of the graph, the outputs are the outputs of the graph, and the levels of the circuit between the product gates and the outputs are the levels of the graph). We would like to prove that

$$s(C) \geq \Omega \left(\frac{1}{d} m^2 \lambda_d(m^2) \right).$$

Assume for a contradiction that $s(C) < \frac{1}{1000d} m^2 \lambda_d(m^2)$ or, in other words,

$$|E_G| < \frac{1}{1000d} m^2 \lambda_d(m^2).$$

By Lemma 3.13, we can find a set of columns D and three sets of vertices V, I, O in the graph G such that the following hold:

- $m \leq |V| = k \leq \frac{1}{10} m^2$.
- I is a small set of inputs.
- D is a set of $\lceil \frac{10k}{m} \rceil$ output columns.
- For every $i \in D$, $|O_i \cap O|$ is small (where O_i is the ith output column).
- All the paths from $O_D \setminus O$ to the inputs pass through V or reach I.

We will derive a contradiction in four steps:

1. Since I is a small set of product gates, we can find a subspace of matrices $\mathcal{A}$ such that, for every matrix X in $\mathcal{A}$, all the gates in I output 0. The matrix X will be restricted to the subspace $\mathcal{A}$.
2. Note that once a matrix X is fixed, the nodes of V just compute linear functions in the variables of Y. Therefore, for every matrix $X \in \mathcal{A}$, we can find a subspace of matrices $\mathcal{B}_X$ such that, for every pair ($X \in \mathcal{A}$, $Y \in \mathcal{B}_X$), all the gates in V output zero. Since V is a small set, the dimension of $\mathcal{B}_X$ is high. The matrix Y will be restricted to the subspace $\mathcal{B}_X$.
3. The subspace $\mathcal{A}$ is of high dimension, and for every X the subspace $\mathcal{B}_X$ is of high dimension. Therefore, we can find $X \in \mathcal{A}$ such that $\dim((X \cdot \mathcal{B}_X)_j)$ is large for some $j \in D$. (This will follow from Lemmas 4.8 and 4.9.)
4. Since we restrict $X \in \mathcal{A}$ and $Y \in \mathcal{B}_X$, all the gates in V and I output zero. Therefore, $X \cdot Y$ is computed by a circuit with no paths between $O_D \setminus O$ and $I_G \setminus I$. Hence all the outputs in $O_D \setminus O$ must give zero. Because of the third step, this is a contradiction.

Let us now give the formal proof.

Proof of Theorem 4.3. We already gave the proof for $d = 1$, so assume $d > 1$. Assume for a contradiction that we have a leveled bilinear arithmetic circuit C of depth d for the product of two $m \times m$ matrices such that

$$s(C) < \frac{1}{1000d} m^2 \lambda_d(m^2).$$

Let G be the leveled graph of depth d between the product gates and the outputs of C (as explained above).

As before, denote by O_j the jth output column. As before, for a set $D \subset [m]$, we denote $O_D = \cup_{i \in D} O_i$. By Lemma 3.13, there exist sets $V \subset V_G$, $D \subset [m]$, $O \subset O_D$, and $I \subset I_G$ such that the following hold:

1. $m \leq |V| \leq \frac{1}{10} m^2$.
2. $|D| \geq \frac{10|V|}{m}$.
3. For every $i \in D$, $|O_i \cap O| < \frac{1}{10} m$.
4. $|I| < \frac{1}{10} m^2$.
5. $P_G[\, I_G \setminus I,\, O_D \setminus O,\, V] = 0$.

Denote $k = |V|$. Hence we have a set of outputs O_D consisting of at least $\frac{10k}{m}$ output columns such that each of the output columns in O_D has a small intersection with O, and there are no paths between $O_D \setminus O$ and $I_G \setminus I$ that do not pass through V.

In the circuit C, the set I is just a set of product gates. Since

$$|I| < \frac{1}{10} m^2,$$

we can find a subspace of matrices $\mathcal{A}$ of dimension

$$\dim(\mathcal{A}) \geq m^2 - |I| > \frac{9}{10}m^2$$

such that, for every $X \in \mathcal{A}$, all the product gates in I give zero.

Assume that a matrix X is fixed. All the functions computed by the vertices of V are now linear functions in the variables of Y. Therefore, for every matrix X, there is a subspace of matrices $\mathcal{B}_X$ such that, for every $Y \in \mathcal{B}_X$, all the gates in V give zero, and such that

$$\dim(\mathcal{B}_X) \geq m^2 - |V| = m^2 - k.$$

Since

$$\dim(\mathcal{A}) \geq \frac{9}{10}m^2,$$

by Lemma 4.9 we can find a matrix $X \in \mathcal{A}$ such that $\operatorname{rank}(X) \geq \frac{2}{3}m$. We fix this X. Since $|D| \geq \frac{10k}{m}$ and $\dim(\mathcal{B}_X) \geq m^2 - k$, we can apply Lemma 4.8 to get a column $j \in D$ such that

$$\dim((X \cdot \mathcal{B}_X)_j) \geq \frac{m}{2}.$$

On the other hand, for $X \in \mathcal{A}$ and $Y \in \mathcal{B}_X$, all the product gates in I and all the gates in V output zero. Since all the paths to $O_D \setminus O$ pass through V or I, we get that, for every $X \in \mathcal{A}$ and $Y \in \mathcal{B}_X$, all the outputs in $O_D \setminus O$ must give zero. Therefore, for every $j \in D$,

$$\dim((X \cdot \mathcal{B}_X)_j) \leq |O_j \cap O| < \frac{m}{10}$$

(a contradiction). □

Theorem 4.3 gives a superlinear lower bound for a multioutput function. The following theorem of [BS82] (it is an immediate corollary of the results presented there) shows that we can also obtain a superlinear lower bound for a single-output function.

THEOREM 4.10 (see [BS82]). *Suppose that $f(x_1, \ldots, x_n)$ is computed by an arithmetic circuit of size s and depth d over a field of characteristic 0; then there is an arithmetic circuit of size $3s$ and depth $2d$ that computes $f, \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}$.*

(Note that in this theorem the size and depth of a circuit are just the usual size and depth; i.e., the size is the number of edges, and the depth is the length of the longest directed path.)

Consider the function

$$f(X, Y, Z) = \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m} x_{i,j}y_{j,k}z_{k,i} = \operatorname{trace}(X \cdot Y \cdot Z),$$

where X, Y, Z are $m \times m$ matrices. Notice that

$$\frac{\partial f}{\partial z_{k,i}} = \sum_{j=1}^{m} x_{i,j}y_{j,k} = (X \cdot Y)_{i,k}.$$

Therefore, the partial derivatives of f with respect to the $Z_{k,i}$'s are the outputs of the product of two matrices. If we take into account the price that we have to pay when transforming a circuit into a bilinear leveled one, we get the following theorem.

THEOREM 4.11. *Every arithmetic circuit C of depth d that computes the trace of the product of three $m \times m$ matrices over a field of characteristic 0 is of size*

$$\Omega\left(\frac{1}{d^2} m^2 \lambda_{6d}(m^2)\right).$$

We can also generalize this result for circuits over finite fields, but we must make another assumption: the circuit computes

$$\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m} x_{i,j} \cdot y_{j,k} \cdot z_{k,i}$$

as a polynomial and not as a function. As mentioned in the introduction, over finite fields there are many polynomials that represent the same function. For example, $x^p - x = 0$ over a field with p elements. Therefore, we demand that the circuit computes this exact polynomial. In this case, we can apply the theorem of Baur and Strassen as before and get the same lower bound as in the case of characteristic 0.

5. Circuits with arbitrary gates. In this section, we prove lower bounds for circuits with arbitrary gates over finite fields. Since the proofs are similar for all finite fields, we will detail only the proofs for the field $GF(2)$, that is, the Boolean case. The proofs for other fields are only sketched.

A Boolean circuit with arbitrary gates is a directed acyclic graph as follows. Nodes of indegree 0 are called inputs and are labeled with input variables. Nodes of outdegree 0 are called outputs. All nodes other than the inputs are labeled with arbitrary Boolean functions. That is, if v is a node of indegree k, then v is labeled with some function

$$g_v : \{0,1\}^k \to \{0,1\}.^1$$

The inputs to the circuit are Boolean variables, and each node in the circuit computes in a natural way a Boolean function in the original input variables. The size of a circuit C is denoted by $size(C)$ and is defined to be the number of edges in it. The depth of a circuit is defined to be the length of the longest directed path from an input to an output in the circuit. Note that the standard definition of a Boolean circuit requires $g_v \in \{\vee, \wedge, \neg\}$. Our definition is more general and allows g_v to be any function.

The model of Boolean circuits with arbitrary gates includes all other models of Boolean circuits, e.g., standard Boolean circuits, Boolean circuits with threshold gates, Boolean circuits with $MODP$ gates, etc. For some of these models, almost nothing is known. For example, for constant depth threshold circuits, only slightly superlinear lower bounds are known [IPS97]. (Exponential lower bounds are known for depth 2 [HMPST87].)

We will mainly concentrate on matrix product over $GF(2)$. Our main result in this section is the following lower bound.

[1] Since g_v is not necessarily a symmetric function, we need to order the inputs to v so we know which input is the first variable of g_v, etc.

THEOREM 5.1. *Any leveled Boolean circuit with arbitrary gates, C, of depth d, for the product of two $m \times m$ Boolean matrices over $GF(2)$, is of size*

$$size(C) = \begin{cases} \Omega(m^3), & d = 1, \\ \Omega(\frac{1}{d} m^2 \lambda_d(m^2)), & d > 1. \end{cases}$$

As before, in order to prove lower bounds for a nonleveled circuit, we just level it. We can do that by increasing its size by a factor of d. We can then use the lower bounds for leveled circuits. We hence have the following corollary.

COROLLARY 5.2. *Any Boolean circuit with arbitrary gates, C, of depth d, for the product of two $m \times m$ Boolean matrices over $GF(2)$, is of size*

$$size(C) = \begin{cases} \Omega(m^3), & d = 1, \\ \Omega(\frac{1}{d^2} m^2 \lambda_d(m^2)), & d > 1. \end{cases}$$

Our proof for these lower bounds is quite general and can be applied to many other functions. In particular, the lower bound applies also for the following functions:

1. The product of two $m \times m$ matrices over $GF(p)$, where each element of the field is represented by $\lceil \log p \rceil$ bits.
2. All the functions considered in [Pud94], e.g., the parallel prefix linear transformation over $GF(2)$.

As mentioned above, we define a similar model of circuits with arbitrary gates over any finite field. We prove similar lower bounds for this model for any finite field. In particular, for the field $GF(p)$, we prove a lower bound for the product of two $m \times m$ matrices, where the inputs take values in the field.

Let us start with a short sketch of the proof of Theorem 5.1. As before, the proof is based on Lemma 3.13. In all that follows, we use the notation of subsection 3.5. We will apply Lemma 3.13 on the circuit C. By Lemma 3.13, there is a set of columns $D \subset [m]$ and small sets of vertices I, O, V such that I is a set of inputs, O is a set of outputs, V is a set of intermediate gates, and

$$P_G[\, I_G \setminus I,\ O_D \setminus O,\ V] = 0$$

(where O_D is the set of outputs corresponding to the set of columns D). The values of the outputs in $O_D \setminus O$ are hence determined by the outputs of the gates in V and by the values of the inputs in I. Therefore, for any fixed assignment for the inputs in I, the total number of possible values that the outputs in $O_D \setminus O$ can take is at most $2^{|V|}$ (which is the total number of values that the gates in V can output). Since V is a small set, $2^{|V|}$ is a relatively small number, and we conclude that, for any fixed assignment for the inputs in I, the outputs in $O_D \setminus O$ can get only a small number of values. We will derive a contradiction by finding a fixed assignment for the inputs in I such that the outputs in $O_D \setminus O$ can get many values.

Let us describe our assignment for I. First, we fix Y to be a matrix in which any minor of size $\frac{m}{2} \times |D|$ is of high rank. Therefore, there will be many vectors in the image of any such minor. After fixing Y, we set to zero all the inputs in I that come from the matrix X. That is, we allow X to be any matrix in which this certain set of entries is zero (the entries that appear in I). Since I is a relatively small set, there are many such matrices. More accurately, the set of all such matrices forms a linear subspace of dimension $\geq m^2 - |I|$. The last step will be to show that, after fixing Y as above and after fixing to 0 all the inputs in I that come from X, there are still many possibilities for the product $X \cdot Y$. In particular, we will get that the number of possible values for the outputs in $O_D \setminus O$ is larger than $2^{|V|}$.

Thus the first step is to show that there is a matrix Y, in which any minor of size $\frac{m}{2} \times |D|$ is of high rank. Note that it is not hard to prove that there exists a matrix Y, in which any such minor is of maximal rank. For our proof, however, it will be enough to have the weaker requirement that any such minor is of high rank.

DEFINITION 5.3. *Let $Y = (y_{i,j})$ be an $m \times m$ matrix. For sets $\alpha, \beta \subset [m]$, denote*

$$(Y)_{\alpha,\beta} = (y_{i,j}) \text{ such that } i \in \alpha,\ j \in \beta;$$

i.e., $(Y)_{\alpha,\beta}$ is the minor of Y with the set of rows α and the set of columns β.

CLAIM 5.4. *For any $m > 40$ and $l \leq \frac{m}{2}$, there exists an $m \times m$ matrix Y (over $GF(2)$) such that, for any $\alpha, \beta \subset [m]$, with $|\alpha| = \lceil \frac{m}{2} \rceil$ and $|\beta| = l$,*

$$\operatorname{rank}((Y)_{\alpha,\beta}) \geq \frac{l}{2}.$$

Proof. We assume for convenience that m, l are even numbers. We will show that a random matrix Y satisfies the requirement of the lemma (with high probability). Let us first calculate the probability that a certain minor of size $\frac{m}{2} \times l$ is of rank $\leq \frac{l}{2}$. As in the proof of Lemma 4.9, the number of $\frac{m}{2} \times l$ matrices of rank $\leq \frac{l}{2}$ is at most

$$\binom{l}{\frac{l}{2}} \cdot 2^{\frac{m}{2} \cdot \frac{l}{2}} \cdot 2^{\frac{l}{2} \cdot (l - \frac{l}{2})} < 2^{\frac{1}{4}ml + \frac{1}{4}l^2 + l}.$$

Therefore, the probability that a certain minor of size $\frac{m}{2} \times l$ is of rank $\leq \frac{l}{2}$ is at most

$$2^{\frac{1}{4}ml + \frac{1}{4}l^2 + l} \cdot 2^{-\frac{m}{2}l} = 2^{-(\frac{1}{4}ml - \frac{1}{4}l^2 - l)}.$$

Hence, if Y is a random matrix, the probability that some $\frac{m}{2} \times l$ minor is of rank $\leq \frac{l}{2}$ is at most

$$\binom{m}{\frac{m}{2}} \cdot \binom{m}{l} \cdot 2^{-(\frac{1}{4}ml - \frac{1}{4}l^2 - l)} \leq 2^{-(\frac{1}{4}ml - \frac{1}{4}l^2 - l - 2m)} < 1.$$

Consequently, there is an $m \times m$ matrix Y, in which every $\frac{m}{2} \times l$ minor is of rank $> \frac{l}{2}$. ∎

For a set of coordinates $\beta \subset [m]$ and an m-vector v, denote by v_β the restriction of v to β; i.e.,

$$v_\beta = (v_i)_{i \in \beta}.$$

For a set of coordinates $\alpha \subset [m]$, denote by $\mathcal{V}_\alpha$ the subspace of all vectors that have the value 0 in all the coordinates outside α; i.e.,

$$\mathcal{V}_\alpha = \{v \in \{0,1\}^m \mid \forall i \in [m] \setminus \alpha,\ v_i = 0\}.$$

We will now prove that, if Y is the matrix guaranteed by Claim 5.4 for $l = |\beta|$, then for every vector space $\mathcal{V}_\alpha$ of high dimension, there are many different vectors in the set

$$\{(v \cdot Y)_\beta \mid v \in \mathcal{V}_\alpha\}.$$

CLAIM 5.5. *Let $\beta \subset [m]$ be such that $|\beta| \leq \frac{m}{2}$. Let $\alpha \subset [m]$ be such that $|\alpha| = \lceil \frac{m}{2} \rceil$. Let Y be an $m \times m$ matrix that satisfies the requirement of Claim 5.4 for*

$l = |\beta|$. Then the number of different vectors of the form $(v \cdot Y)_\beta$, where $v \in \mathcal{V}_\alpha$, is at least $2^{\frac{|\beta|}{2}}$.

Proof. Since Y satisfies the requirement of Claim 5.4 for $l = |\beta|$, the minor $(Y)_{\alpha,\beta}$ is of rank at least $|\beta|/2$. Hence the image of this minor is of dimension at least $|\beta|/2$. The image of this minor is just the set of all vectors of the form $(v \cdot Y)_\beta$, where $v \in \mathcal{V}_\alpha$. Hence this set is of size at least $2^{\frac{|\beta|}{2}}$. □

We are now ready to give the proof of Theorem 5.1.

Proof of Theorem 5.1. For $d = 1$, the proof is trivial by the observation that every output column depends on all the variables in X. For larger depth, assume for a contradiction that $size(C) < (1/1000d) \cdot m^2 \lambda_d(m^2)$. Let G be the graph of the circuit. By Lemma 3.13, there exist sets $V \subset V_G$, $D \subset [m]$, $O \subset O_D$, $I \subset I_G$, such that the following hold:

1. $m \leq |V| = k \leq \frac{1}{30}m^2$.
2. $|D| = \lceil \frac{10k}{m} \rceil < \frac{m}{2}$.
3. For every $i \in D$, $|O_i \cap O| < \frac{1}{10}m$.
4. $|I| < \frac{1}{10}m^2$.
5. $P_G[\, I_G \setminus I,\, O_D \setminus O,\, V] = 0$.

Since $P_G[\, I_G \setminus I,\, O_D \setminus O,\, V] = 0$, the values of the outputs in $O_D \setminus O$ are determined by the outputs of the gates in V and by the inputs in I.

Fix Y to be a matrix in which every $\lceil \frac{m}{2} \rceil \times |D|$ minor is of rank higher than $\frac{|D|}{2}$. (By Claim 5.4, there exists such a matrix.) Denote by I_X the set of inputs in I that are variables of X. Obviously,

$$|I_X| \leq |I| < \frac{1}{10}m^2.$$

Fix all the input variables in I_X to be 0. Since all the inputs in I are now fixed, the values of the outputs in $O_D \setminus O$ are determined by the outputs of the gates in V. Since there are at most 2^k possible values for the outputs of the gates in V, we conclude that, after fixing Y and I_X as above, the outputs in $O_D \setminus O$ can get at most 2^k different values.

On the other hand, we fixed only less than $\frac{1}{10}m^2$ of the entries of X. Therefore, the number of rows of X, in which we fixed at most $\frac{m}{2}$ entries, is at least $\frac{8}{10}m$. For each one of these rows, we can apply Claim 5.5 (with $\beta = D$) and conclude that the outputs in the corresponding row in O_D can get at least $2^{\frac{|D|}{2}}$ possible values. Since the value of each of these $\frac{8}{10}m$ rows of X is independent of the values of the other rows, we conclude that the total number of different values that the outputs in O_D can get is at least

$$2^{\frac{|D|}{2} \cdot \frac{8}{10}m} = 2^{\frac{4}{10} \cdot |D| \cdot m}.$$

Since, for every $i \in D$, $|O_i \cap O| < \frac{1}{10}m$, we get that

$$|O| < \frac{1}{10} \cdot |D| \cdot m.$$

Hence the outputs in O can get at most

$$2^{\frac{1}{10} \cdot |D| \cdot m}$$

different values. Therefore, the outputs in $O_D \setminus O$ can get at least

$$2^{\frac{4}{10} \cdot |D| \cdot m} \,/\, 2^{\frac{1}{10} \cdot |D| \cdot m} = 2^{\frac{3}{10} \cdot |D| \cdot m} \geq 2^{3k}$$

different values (a contradiction). □

As mentioned above, we can also obtain similar lower bounds for circuits with arbitrary gates over any finite field. A circuit with arbitrary gates over a finite field $GF(p)$ is defined similarly to a circuit with arbitrary gates over $GF(2)$. The only difference is that the inputs take values in $GF(p)$, and every gate of indegree k is labeled with an arbitrary function from $GF(p)^k$ to $GF(p)$. Note that, in particular, this model includes the model of arithmetic circuits over $GF(p)$.

THEOREM 5.6. *Any circuit C with arbitrary gates over $GF(p)$, where p is constant, of depth d for the product of two $m \times m$ matrices over $GF(p)$ is of size*

$$size(C) = \begin{cases} \Omega(m^3), & d = 1, \\ \Omega(\frac{1}{d^2} m^2 \lambda_d(m^2)), & d > 1. \end{cases}$$

The proof is similar to the proof of Theorem 5.1 with some minor modifications: We prove a version of Claim 5.4 to get a matrix with the same properties over $GF(p)$. Then we prove a version of Claim 5.5 for $GF(p)$. (Both proofs are similar to the original proofs.) We then repeat the proof of Theorem 5.1 to get Theorem 5.6. (We have to make some minor changes; e.g., the outputs of the gates in V can get $p^{|V|}$ different values (rather than $2^{|V|}$), etc.)

We can also prove similar lower bounds for Boolean circuits with arbitrary gates for the product of two $m \times m$ matrices over the field $GF(p)$, where each element of the field is represented by $\lceil \log p \rceil$ bits. (Note that the input to the circuit is of size $2m^2 \lceil \log p \rceil$.) One way of proving this lower bound is by a reduction to Theorem 5.6. Just observe that $\{0, 1\} \subset GF(p)$, and hence any Boolean circuit (with arbitrary gates) can be viewed as a circuit with arbitrary gates over $GF(p)$. Another way of proving this lower bound is by proving a version of Lemma 3.13 with different parameters (because of the $\lceil \log p \rceil$ factor), and then we can repeat the proof of Theorem 5.1 with minor modifications.

THEOREM 5.7. *Any Boolean circuit with arbitrary gates, C, of depth d, for the product of two $m \times m$ matrices over $GF(p)$ (p is constant), is of size*

$$size(C) = \begin{cases} \Omega(m^3), & d = 1, \\ \Omega(\frac{1}{d^2} m^2 \lambda_d(m^2)), & d > 1. \end{cases}$$

Acknowledgments. We would like to thank Toni Pitassi and Avi Wigderson for helpful conversation.

REFERENCES

[Ajt83] M. AJTAI, Σ_1^1-*formulae on finite structures*, Ann. Pure Appl. Logic, 24 (1983), pp. 1–48.

[Bla99] M. BLASER, *A $\frac{5}{2}n^2$-lower bound for the rank of $n \times n$-matrix multiplication over arbitrary fields*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 45–50.

[Bsh89] N. H. BSHOUTY, *A lower bound for matrix multiplication*, SIAM J. Comput., 18 (1989), pp. 759–765.

[BS82] W. BAUR AND V. STRASSEN, *The complexity of partial derivatives*, Theoret. Comput. Sci., 22 (1982), pp. 317–330.

[DDPW83] D. DOLEV, C. DWORK, N. J. PIPPENGER, AND A. WIGDERSON, *Superconcentrators, generalizer and generalized connectors*, in Proceedings of the 15th Annual ACM Symposium on Theory of Computing, ACM, New York, 1983, pp. 42–51.

[FSS81] M. L. Furst, J. B. Saxe, and M. Sipser, *Parity, circuits, and the polynomial-time hierarchy*, in Proceedings of the 22nd Annual IEEE Synposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1981, pp. 260–270.

[Gat88] J. von zur Gathen, *Algebraic complexity theory*, in Annual Review of Computer Science, Vol. 3, Annual Reviews, Palo Alto, CA, 1988, pp. 317–347.

[GK98] D. Grigoriev and M. Karpinski, *An exponential lower bound for depth* 3 *arithmetic circuits*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 577–582.

[GR98] D. Grigoriev and A. A. Razborov, *Exponential complexity lower bounds for depth* 3 *arithmetic circuits in algebras of functions over finite fields*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1998, pp. 269–278.

[Hås86] J. Håstad, *Almost optimal lower bounds for small depth circuits*, in Proceedings of the 18th Annual ACM Symposium on Theory of Computing, ACM, New York, 1986, pp. 6–20.

[HMPST87] A. Hajnal, W. Maass, P. Pudlak, M. Szegedy, and G. Turan, *Threshold circuits of bounded depth*, in Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1987, pp. 99–110.

[IPS97] R. Impagliazzo, R. Paturi, and M. E. Saks, *Size–depth tradeoffs for threshold circuits*, SIAM J. Comput., 26 (1997), pp. 693–707.

[Pud94] P. Pudlak, *Communication in bounded depth circuits*, Combinatorica, 14 (1994), pp. 203–216.

[Razb87] A. A. Razborov, *Lower bounds for the size of circuits with bounded depth with basis* $\{\wedge, \oplus\}$, Math. Notes of the Academy of Science of the USSR, 41 (1987), pp. 333–338.

[Smo87] R. Smolensky, *Algebraic methods in the theory of lower bounds for Boolean circuit complexity*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, ACM, New York, 1987, pp. 77–82.

[Str69] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[Str73] V. Strassen, *Die berechnungskomplexitat von elementarsymmetrischen funktionen und von interpolationskoefizienten*, Numer. Math., 20 (1973), pp. 238–251.

[SW99] A. Shpilka and A. Wigderson, *Depth-*3 *arithmetic formulae over fields of characteristic zero*, Electronic Colloquium on Computational Complexity, 6(023), 1999.

[Val77] L. G. Valiant, *Graph-theoretic arguments in low-level complexity*, in Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 53, Springer-Verlag, Berlin, 1977, pp. 162–176.

[Yao85] A. C. Yao, *Separating the polynomial hierarchy by oracles*, in Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1985, pp. 1–10.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 514–537

NEW RESULTS ON MONOTONE DUALIZATION AND GENERATING HYPERGRAPH TRANSVERSALS*

THOMAS EITER†, GEORG GOTTLOB†, AND KAZUHISA MAKINO‡

Abstract. We consider the problem of dualizing a monotone CNF (equivalently, computing all minimal transversals of a hypergraph) whose associated decision problem is a prominent open problem in NP-completeness. We present a number of new polynomial time, respectively, output-polynomial time results for significant cases, which largely advance the tractability frontier and improve on previous results. Furthermore, we show that duality of two monotone CNFs can be disproved with limited nondeterminism. More precisely, this is feasible in polynomial time with $O(\log^2 n/\log\log n)$ suitably guessed bits. This result sheds new light on the complexity of this important problem.

Key words. dualization, hypergraphs, transversal computation, output-polynomial algorithms, combinatorial enumeration, treewidth, hypergraph acyclicity, limited nondeterminism

AMS subject classifications. 05C65, 05C85, 05C90, 06E30, 68P15, 68Q20, 68Q25, 68R10, 68T30, 94C10

PII. S009753970240639X

1. Introduction.

Recall that the prime conjunctive normal form (CNF) of a monotone Boolean function f is the unique formula $\varphi = \bigwedge_{c\in S} c$ in conjunctive normal form, where S is the set of all prime implicates of f, i.e., minimal clauses c which are logical consequences of f. In this paper, we consider the following problem.

Problem DUALIZATION
Input: The prime CNF φ of a monotone Boolean function $f = f(x_1, \ldots, x_m)$.
Output: The prime CNF ψ of its dual $f^a d = \overline{f}(\overline{x}_1, \ldots, \overline{x}_m)$.

It is well known that DUALIZATION is equivalent to the TRANSVERSAL COMPUTATION problem, which requests to compute the set of all minimal transversals (i.e., minimal hitting sets) of a given hypergraph $\mathcal{H}$, in other words, the *transversal hypergraph* $Tr(\mathcal{H})$ of $\mathcal{H}$. Actually, these problems can be viewed as the same problem if the clauses in a monotone CNF φ are identified with the sets of variables they contain. DUALIZATION is a search problem; the associated decision problem DUAL is to decide whether two given monotone prime CNFs φ and ψ represent a pair (f, g) of dual Boolean functions. Analogously, the decision problem TRANS-HYP associated with TRANSVERSAL COMPUTATION is deciding, given hypergraphs $\mathcal{H}$ and $\mathcal{G}$, whether $\mathcal{G} = Tr(\mathcal{H})$.

DUALIZATION and several problems which are like transversal computation known to be computationally equivalent to problem DUALIZATION (see [15]) are of interest

*Received by the editors April 25, 2002; accepted for publication (in revised form) October 28, 2002; published electronically February 20, 2003. A shorter version of this paper appeared in *Proceedings of the 34th ACM Symposium on Theory of Computing*, Montreal, Quebec, Canada, 2002, pp. 14–22. This work was supported in part by the Austrian Science Fund (FWF) project Z29-INF, by TU Wien through a scientific collaboration grant, and by the Scientific Grant in Aid of the Ministry of Education, Culture, Sports, Science and Technology of Japan.

http://www.siam.org/journals/sicomp/32-2/40639.html

†Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Vienna, Austria (eiter@kr.tuwien.ac.at, gottlob@dbai.tuwien.ac.at).

‡Division of Systems Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka, 560-8531, Japan (makino@sys.es.osaka-u.ac.jp).

in various areas such as database theory (e.g., [39, 50]), machine learning and data mining (e.g., [7, 8, 10, 25]), game theory (e.g., [26, 43, 44]), artificial intelligence (e.g., [21, 30, 32, 45]), mathematical programming (e.g., [5]), and distributed systems (e.g., [18, 27]), to mention a few.

While the output CNF ψ can be exponential in the size of φ, it is currently not known whether ψ can be computed in *output-polynomial* (or *polynomial total*) *time*, i.e., in time polynomial in the combined size of φ *and* ψ. Any such algorithm for DUALIZATION (or for TRANSVERSAL COMPUTATION) would significantly advance the state of the art of several problems in the above application areas. Similarly, the complexity of DUAL (equivalently, TRANS-HYP) has been open for more than 20 years now (cf. [3, 15, 28, 29, 34]).

Note that DUALIZATION is solvable in polynomial total time on a class $\mathcal{C}$ of hypergraphs iff DUAL is in PTIME for all pairs $(\mathcal{H}, \mathcal{G})$, where $\mathcal{H} \in \mathcal{C}$ [3]. DUAL is known to be in co-NP, and the best currently known upper time-bound is quasi-polynomial time [17, 19, 48]. Determining the complexities of DUALIZATION and DUAL, and of equivalent problems such as the transversal problems, is a prominent open problem. This is witnessed by the fact that these problems are cited in a rapidly growing body of literature and have been referenced in various survey papers and complexity theory retrospectives, e.g., [28, 35, 40].

Given the importance of monotone dualization and equivalent problems for many application areas, and given the long-standing failure to settle the complexity of these problems, emphasis was put on finding tractable cases of DUAL and corresponding polynomial total time cases of DUALIZATION. In fact, several relevant tractable classes were found by various authors; see, e.g., [4, 6, 9, 12, 10, 14, 15, 20, 37, 38, 42, 41] and references therein. Moreover, classes of formulas were identified on which DUALIZATION is not just polynomial total time, but where the conjuncts of the dual formula can be enumerated with *incremental polynomial delay*, i.e., with delay polynomial in the size of the input plus the size of all conjuncts so far computed, or even with *polynomial delay*, i.e., with delay polynomial in the input size only. On the other hand, there are also results which show that certain well-known algorithms for DUALIZATION are not polynomial total time. For example, [15, 42] pointed out that a well-known sequential algorithm, in which the clauses c_i of a CNF $\varphi = c_1 \wedge \cdots \wedge c_m$ are processed in order $i = 1, \ldots, m$, is not polynomial total time in general. Most recently, [47] showed that this holds even if an optimal ordering of the clauses is assumed (i.e., they may be arbitrarily arranged for free).

Main goal. The main goal of this paper is to present important new polynomial total time cases of DUALIZATION and, correspondingly, PTIME solvable subclasses of DUAL which significantly improve previously considered classes. Toward this aim, we first present a new algorithm DUALIZE and prove its correctness. DUALIZE can be regarded as a generalization of a related algorithm proposed by Johnson, Yannakakis, and Papadimitriou [29]. Like other dualization algorithms, DUALIZE reduces the original problem by self-reduction to smaller instances. However, the subdivision into subproblems proceeds according to a particular order, which is induced by an arbitrary fixed ordering of the variables. This, in turn, allows us to derive some bounds on intermediate computation steps which imply that DUALIZE, when applied to a variety of input classes, outputs the conjuncts of ψ with polynomial delay or incremental polynomial delay. In particular, we show positive results for the following input classes.

Degenerate CNFs. We generalize the notion of k-degenerate graphs [51] to hypergraphs and define *k-degenerate monotone CNFs*, respectively, *hypergraphs*. A mono-

tone CNF is *k-degenerate* if there exists a variable ordering $x_1, \ldots, x_n$ such that, for $i = 1, 2, \ldots, n$, the number of clauses which contain x_i and, apart from it, only variables from $x_1, \ldots, x_{i-1}$ is at most k. We prove that, for any constant k, DUALIZE works with polynomial delay on k-degenerate CNFs. Moreover, it works in output-polynomial time on $O(\log n)$-degenerate CNFs.

Read-k CNFs. A CNF is *read-k* if each variable appears at most k times in it. We show that, for read-k CNFs, problem DUALIZATION is solvable with polynomial delay if k is constant and in total polynomial time if $k = O(\log \|\varphi\|)$. Our result for constant k significantly improves upon the previous best-known algorithm [10], which has a higher complexity bound, is not polynomial delay, and outputs the clauses of ψ in no specific order. The result for $k = O(\log \|\varphi\|)$ is a nontrivial generalization of the result in [10], which was posed as an open problem [13].

Acyclic CNFs. There are several notions of hypergraph, respectively, monotone CNF acyclicity [16], where the most general and well-known is α-acyclicity. As shown in [15], DUALIZATION is polynomial total time for β-acyclic CNFs; β-acyclicity is the hereditary version of α-acyclicity and far less general. A similar result for α-acyclic prime CNFs was left open. (For nonprime α-acyclic CNFs, this is trivially as hard as the general case.) In this paper, we give a positive answer and show that, for α-acyclic (prime) φ, DUALIZATION is solvable with polynomial delay.

Formulas of bounded treewidth. The *treewidth* [46] of a graph expresses its degree of cyclicity. Treewidth is an extremely general notion, and bounded treewidth generalizes almost all other notions of near-acyclicity. Following [11], we define the treewidth of a hypergraph, respectively, monotone CNF φ, as the treewidth of its associated (bipartite) variable-clause incidence graph. We show that DUALIZATION is solvable with polynomial delay (exponential in k) if the treewidth of φ is bounded by a constant k and in polynomial total time if the treewidth is $O(\log \log \|\varphi\|)$.

Recursive applications of DUALIZE *and k-CNFs.* We show that if DUALIZE is applied recursively and the recursion depth is bounded by a constant, then DUALIZATION is solved in polynomial total time. We apply this to provide a simpler proof of the known result [6, 15] that monotone k-CNFs (where each conjunct contains at most k variables) can be dualized in output-polynomial time.

After deriving the above results, we turn our attention in section 5 to the fundamental computational nature of problems DUAL and TRANS-HYP in terms of complexity theory.

Limited nondeterminism. In a landmark paper, Fredman and Khachiyan [17] proved that problem DUAL can be solved in quasi-polynomial time. More precisely, they first gave an Algorithm A solving the problem in $n^{O(\log^2 n)}$ time and then a more complicated Algorithm B whose runtime is bounded by $n^{4\chi(n)+O(1)}$, where $\chi(n)$ is defined by $\chi(n)^{\chi(n)} = n$. As noted in [17], $\chi(n) \sim \log n / \log \log n = o(\log n)$; therefore, duality checking is feasible in $n^{o(\log n)}$ time. This is the best upper bound for problem DUAL so far and shows that the problem is most likely not NP-complete.

A natural question is whether DUAL lies in some lower complexity class based on other resources than just runtime. In the present paper, we advance the complexity status of this problem by showing that its complement is feasible with *limited nondeterminism*, i.e, by a nondeterministic polynomial time algorithm that makes only a polylogarithmic number of guesses. For a survey on complexity classes with limited nondeterminism and for several references, see [22]. We first show by using a simple but effective technique, which succinctly describes computation paths, that testing nonduality is feasible in polynomial time with $O(\log^3 n)$ nondeterministic steps. We then observe that this approach can be improved to obtain a bound of

$O(\chi(n) \cdot \log n) = O(\log^2 n / \log\log n)$ nondeterministic steps. *This result is surprising because most researchers dealing with the complexity of* DUAL *and* TRANS-HYP *believed so far that these problems are completely unrelated to limited nondeterminism.*

We believe that the results presented in this paper are significant, and we are confident that they will be proven useful in various contexts. First, we hope that the various polynomial/output-polynomial cases of the problems which we identify will lead to better and more general methods in various application areas (as we show, e.g., in learning and data mining [10]) and that, based on the algorithm DUALIZE or some future modifications, further relevant tractable classes will be identified. Second, we hope that our discovery on limited nondeterminism will provide a new momentum to complexity research on DUAL and TRANS-HYP and will push toward settling these long-standing open problems.

The rest of this paper is structured as follows. The next section provides some preliminaries and introduces notation. In section 3, we present our Algorithm DUALIZE for dualizing a given monotone prime CNF. After that, we exploit this algorithm in section 4 to derive a number of polynomial instance classes of the problems DUALIZATION and DUAL. In section 5, we then show that DUAL can be solved with limited nondeterminism.

2. Preliminaries and notation. A *Boolean function* (in short, *function*) is a mapping $f : \{0,1\}^n \to \{0,1\}$, where $v \in \{0,1\}^n$ is called a *Boolean vector* (in short, *vector*). As usual, we write $g \leq f$ if f and g satisfy $g(v) \leq f(v)$ for all $v \in \{0,1\}^n$, and $g < f$ if $g \leq f$ and $g \neq f$. A function f is *monotone* (or *positive*) if $v \leq w$ (i.e., $v_i \leq w_i$ for all i) implies $f(v) \leq f(w)$ for all $v, w \in \{0,1\}^n$. Boolean variables $x_1, x_2, \ldots, x_n$ and their complements $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$ are called *literals*. A *clause* (resp., *term*) is a disjunction (resp., conjunction) of literals containing at most one of x_i and $\bar{x}_i$ for each variable. A clause c (resp., term t) is an *implicate* (resp., *implicant*) of a function f if $f \leq c$ (resp., $t \leq f$); moreover, it is *prime* if there is no implicate $c' < c$ (resp., no implicant $t' > t$) of f and *monotone* if it consists only of positive literals. We denote by $PI(f)$ the set of all prime implicants of f.

A *conjunctive normal form* (CNF) (resp., disjunctive normal form (DNF)) is a conjunction of clauses (resp., disjunction of terms); it is *prime* (resp., *monotone*) if all its members are prime (resp., *monotone*). For any CNF (resp., DNF) ρ, we denote by $|\rho|$ the number of clauses (resp., terms) in it. Furthermore, for any formula φ, we denote by $V(\varphi)$ the set of variables that occur in φ and by $\|\varphi\|$ its *length*, i.e., the number of literals in it. We occasionally view CNFs φ also as sets of clauses, and clauses as sets of literals, and use respective notation (e.g., $c \in \varphi$, $\bar{x}_1 \in c$, etc.).

As is well known, a function f is monotone iff it has a monotone CNF. Furthermore, all prime implicants and prime implicates of a monotone f are monotone, and it has a unique prime CNF, given by the conjunction of all its prime implicates. For example, the monotone f such that $f(v) = 1$ iff $v \in \{(1100), (1110), (1101), (0111), (1111)\}$ has the unique prime CNF $\varphi = x_2(x_1 \vee x_3)(x_1 \vee x_4)$.

Recall that the *dual* of a function f, denoted f^d, is defined by $f^d(x) = \overline{f}(\overline{x})$, where $\overline{f}$ and $\overline{x}$ are the complements of f and x, respectively. By definition, we have $(f^d)^d = f$. From De Morgan's law, we obtain a formula for f^d from any one of f by exchanging $\vee$ and $\wedge$ as well as the constants 0 and 1. For example, if f is given by $\varphi = x_1x_2 \vee \bar{x}_1(\bar{x}_3 \vee x_4)$, then f^d is represented by $\psi = (x_1 \vee x_2)(\bar{x}_1 \vee \bar{x}_3x_4)$. For a monotone function f, let $\psi = \bigwedge_{c \in C}(\bigvee_{x_i \in c} x_i)$ be the prime CNF of f^d. Then, by De Morgan's law, f has the (unique) prime DNF $\rho = \bigvee_{c \in C}(\bigwedge_{x_i \in c} x_i)$; in the previous example, $\rho = x_1x_2 \vee x_2x_3x_4$. Thus we will regard DUALIZATION also as the problem

of computing the prime DNF of f from the prime CNF of f.

3. Ordered transversal generation. In what follows, let f be a monotone function, and let

$$\varphi = \bigwedge_{i=1}^{m} c_i \tag{1}$$

be a monotone CNF of it, where we assume without loss of generality that all variables x_j $(j = 1, 2, \ldots, n)$ appear in φ. Let φ_i $(i = 0, 1, \ldots, n)$ be the CNF obtained from φ by fixing variables $x_j = 1$ for all j with $j \geq i+1$. By definition, we have $\varphi_0 = 1$ (truth) and $\varphi_n = \varphi$. For example, consider $\varphi = (x_1 \vee x_2)(x_1 \vee x_3)(x_2 \vee x_3 \vee x_4)(x_1 \vee x_4)$. Then we have $\varphi_0 = \varphi_1 = 1$, $\varphi_2 = (x_1 \vee x_2)$, $\varphi_3 = (x_1 \vee x_2)(x_1 \vee x_3)$, and $\varphi_4 = \varphi$. Similarly, for a monotone DNF

$$\psi = \bigvee_{t=1}^{k} t_i \tag{2}$$

of f, we denote by ψ_i the DNF obtained from ψ by fixing variables $x_j = 1$ for all j with $j \geq i + 1$. Clearly, we have $\varphi_i \equiv \psi_i$; i.e., φ_i and ψ_i represent the same function denoted by f_i.

PROPOSITION 3.1. *Let φ and ψ be any CNF and DNF for f, respectively. Then, for all $i \geq 0$,* (a) $\|\varphi_i\| \leq \|\varphi\|$ *and* $|\varphi_i| \leq |\varphi|$, *and* (b) $\|\psi_i\| \leq \|\psi\|$ *and* $|\psi_i| \leq |\psi|$.

Denote by Δ^i $(i = 1, 2, \ldots, n)$ the CNF consisting of all the clauses in φ_i but not in φ_{i-1}. For the above example, we have $\Delta^1 = 1$, $\Delta^2 = (x_1 \vee x_2)$, $\Delta^3 = (x_1 \vee x_3)$, and $\Delta^4 = (x_2 \vee x_3 \vee x_4)(x_1 \vee x_4)$. Note that $\varphi_i = \varphi_{i-1} \wedge \Delta^i$; hence, for all $i = 1, 2, \ldots, n$, we have

$$\psi_i \equiv \psi_{i-1} \wedge \Delta^i \equiv \bigvee_{t \in PI(f_{i-1})} (t \wedge \Delta^i). \tag{3}$$

Remark 3.1. Let Γ^i denote the prime DNF for Δ^i. For k-degenerate monotone CNFs, the inequalities $|\psi_i| \leq |\psi|$ and $|\Gamma^i| \leq n^k$ readily imply in light of (3) that such CNFs can be dualized in output-polynomial time by simply multiplying the clauses in all Δ^i and combining them in the order specified by their highest-rank variables. However, we shall present a faster algorithm.

Let $\Delta^i[t]$ for $i = 1, \ldots, n$ denote the CNF consisting of all the clauses c such that c contains no literal in t_{i-1} and $c \vee x_i$ appears in Δ^i. For example, if $t = x_2x_3x_4$ and $\Delta^4 = (x_2 \vee x_3 \vee x_4)(x_1 \vee x_4)$, then $\Delta^4[t] = x_1$. It follows from (3) that, for all $i = 1, 2, \ldots, n$,

$$\psi_i \equiv \bigvee_{t \in PI(f_{i-1})} \big((t \wedge \Delta^i[t]) \ \vee \ (t \wedge x_i)\big). \tag{4}$$

In what follows, let φ and ψ be the prime CNF and prime DNF of f, respectively.

LEMMA 3.2. *For every term $t \in PI(f_{i-1})$, let $g_{i,t}$ be the function represented by $\Delta^i[t]$. Then $|PI(g_{i,t})| \leq |\psi_i| \leq |\psi|$.*

Proof. Let $V = \{x_1, x_2, \ldots, x_n\}$, and let $s \in PI(g_{i,t})$. Then, by (4), $t \wedge s$ is an implicant of ψ_i. Hence some $t^s \in PI(f_i)$ exists such that $t^s \geq t \wedge s$. Note that $V(t) \cap V(\Delta^i[t]) = \emptyset$, t and $\Delta^i[t]$ have no variable in common, and hence we have $V(s) \subseteq V(t^s)\,(\subseteq V(s) \cup V(t))$, since otherwise there exists a clause c in $\Delta^i[t]$ such

that $V(c) \cap V(t^s) = \emptyset$, which is a contradiction. Thus $V(t^s) \cap V(\Delta^i[t]) = V(s)$. For any $s' \in PI(g_{i,t})$ such that $s \neq s'$, let $t^s, t^{s'} \in PI(f_i)$ such that $t^s \geq t \wedge s$ and $t^{s'} \geq t \wedge s'$, respectively. By the above discussion, we have $t^s \neq t^{s'}$. This completes the proof. □

We now describe our Algorithm DUALIZE for generating $PI(f)$. It is inspired by a similar graph algorithm of Johnson, Yannakakis, and Papadimitriou [29] and can be regarded as a generalization.

Algorithm DUALIZE

Input: The prime CNF φ of a monotone function f.
Output: The prime DNF ψ of f, i.e., all prime implicants of f.

Step 1:

compute the smallest prime implicant t_{min} of f and set $Q := \{ t_{min} \}$;

Step 2:

while $Q \neq \emptyset$ **do begin**
  remove the smallest t from Q and output t;
  for each i with $x_i \in V(t)$ and $\Delta^i[t] \neq 1$ **do begin**
    compute the prime DNF $\rho_{(t,i)}$ of the function represented by $\Delta^i[t]$;
    for each term t' in $\rho_{(t,i)}$ **do begin**
      if $t_{i-1} \wedge t'$ is a prime implicant of f_i **then begin**
        compute the smallest prime implicant t^* of f such that $t_i^* = t_{i-1} \wedge t'$;
        $Q := Q \cup \{t^*\}$
      end{if}
    end{for}
  end{for}
end{while}

Here, we say that term s is *smaller* than term t if $\sum_{x_j \in V(s)} 2^{n-j} < \sum_{x_j \in V(t)} 2^{n-j}$; i.e., as vector, s is lexicographically smaller than t.

THEOREM 3.3. *Algorithm* DUALIZE *correctly outputs all* $t \in PI(f)$ *in increasing order.*

Proof. First note that the term t^* inserted in Q when t is output is larger than t. Indeed, t' $(\neq 1)$ and t_{i-1} are disjoint and $V(t') \subseteq \{x_1, \ldots, x_{i-1}\}$. Hence every term in Q is larger than all terms already output, and the output sequence is increasing. We show by induction that, if t is the smallest prime implicant of f that was not output yet, then t is already in Q. This clearly proves the result.

Clearly, the above statement is true if $t = t_{min}$. Assume now that $t \neq t_{min}$ is the smallest among the prime implicants not output yet. Let i be the largest index such that t_i is not a prime implicant of f_i. This i is well defined since otherwise $t = t_{min}$ must hold, which is a contradiction. Now we have (1) $i < n$ and (2) $i+1 \notin V(t)$, where (1) holds because t_n $(= t)$ is a prime implicant of f_n $(= f)$ and (2) follows from the maximality of i. Let $s \in PI(f_i)$ such that $V(s) \subseteq V(t_i)$, and let $K = V(t_i) - V(s)$. Then $K \neq \emptyset$ holds, and since $x_{i+1} \notin V(t)$, the term $t' = \bigwedge_{x_j \in K} x_j$ is a prime implicant of $\Delta^{i+1}[s]$. There exists $s' \in PI(f)$ such that $s'_i = s$ and $x_{i+1} \in V(s')$ since $s \wedge x_{i+1} \in PI(f_{i+1})$. Note that $\Delta^{i+1}[s] \neq 0$. Moreover, since s' is smaller than t, by induction s' has already been output. Therefore, $t' = \bigwedge_{x_j \in K} x_j$ has been considered in the inner for-loop of the algorithm. Since $s'_i \wedge t'$ $(= t_i = t_{i+1})$ is a prime implicant of f_{i+1}, the algorithm has added the smallest prime implicant t^* of f such that $t^*_{i+1} = t_{i+1}$. We finally claim that $t^* = t$. Otherwise, let k be the first index

in which t^* and t differ. Then $k > i+1$, $x_k \in V(t)$, and $x_k \notin V(t^*)$. However, this implies $t_k \notin PI(f_k)$, contradicting the maximality of i. □

Remark 3.2. (1) The decomposition rule (4) was already used in [34]. (2) In Step 1, we could generate any prime implicant t of f and choose then a lexicographic term ordering inherited from a dynamically generated variable ordering. In Step 2, it is sufficient that any monotone DNF $\tau_{(t,i)}$ of the function represented by $\Delta^i[t]$ is computed, rather than its prime DNF $\rho_{(t,i)}$. This might make the algorithm faster.

Let us consider the time complexity of Algorithm DUALIZE. We store Q as a binary tree, where each leaf represents a term t and the left (resp., right) son of a node at depth $j-1 \geq 0$, where the root has depth 0 and encodes $x_j \in V(t)$ (resp., $x_j \notin V(t)$). In Step 1, we can compute t_{min} in $O(\|\varphi\|)$ time and initialize Q in $O(n)$ time.

As for Step 2, let $T_{(t,i)}$ be the time required to compute the prime DNF $\rho_{(t,i)}$ from $\Delta^i[t]$. By analyzing its substeps, we can see that each iteration of Step 2 requires $\sum_{x_i \in V(t)}(T_{(t,i)} + |\rho_{(t,i)}| \cdot O(\|\varphi\|))$ time.

Indeed, we can update Q (i.e., remove the smallest term and add t^*) in $O(n)$ time. For each t and i, we can construct $\Delta^i[t]$ in $O(\|\varphi\|)$ time. Moreover, we can check whether $t_{i-1} \wedge t'$ is a prime implicant of f_i, and, if so, we can compute the smallest prime implicant t^* of f such that $t_i^* = t_{i-1} \wedge t'$ in $O(\|\varphi\|)$ time; note that t^* is the smallest prime implicant of the function obtained from f by fixing $x_j = 1$ if $x_j \in V(t_i \wedge t')$ and 0 if $x_j \notin V(t_i \wedge t')$ for $j \leq i$.

Hence we have the following result.

THEOREM 3.4. *The output delay of Algorithm* DUALIZE *is bounded by*

$$\max_{t \in PI(f)} \left(\sum_{x_i \in V(t)} (T_{(t,i)} + |\rho_{(t,i)}| \cdot O(\|\varphi\|)) \right) \tag{5}$$

time, and DUALIZE *needs in total time*

$$\sum_{t \in PI(f)} \sum_{x_i \in V(t)} (T_{(t,i)} + |\rho_{(t,i)}| \cdot O(\|\varphi\|)). \tag{6}$$

If the $T_{(t,i)}$ are bounded by a polynomial in the input length, then DUALIZE becomes a polynomial delay algorithm since $|\rho_{(t,i)}| \leq T_{(t,i)}$ holds for all $t \in PI(f)$ and $x_i \in V(t)$. On the other hand, if they are bounded by a polynomial in the combined input and output length, then DUALIZE is a polynomial total time algorithm, where $|\rho_{(t,i)}| \leq |\psi|$ holds from Lemma 3.2. Using results from [3], we can construct from DUALIZE an incremental polynomial time algorithm for DUALIZATION, which, however, might not output $PI(f)$ in increasing order. Summarizing, we have the following corollary.

COROLLARY 3.5. *Let* $T = \max\{T_{(t,i)} \mid t \in PI(f), x_i \in V(t)\}$. *Then,*

(i) *if* T *is bounded by a polynomial in* n *and* $\|\varphi\|$, *then* DUALIZE *is an* $O(n\|\varphi\|T)$ *polynomial delay algorithm;*

(ii) *if* T *is bounded by a polynomial in* n, $\|\varphi\|$, *and* $\|\psi\|$, *then* DUALIZE *is an* $O(n \cdot |\psi| \cdot (T + |\psi| \cdot \|\varphi\|))$ *polynomial total time algorithm; moreover,* DUALIZATION *is solvable in incremental polynomial time.*

In the next section, we identify sufficient conditions for the boundedness of T and fruitfully apply them to solve open problems and improve previous results.

4. Polynomial classes.

4.1. Degenerate CNFs. We first consider the case of small $\Delta^i[t]$. Generalizing a notion for graphs (i.e., monotone 2-CNFs) [51], we call a monotone CNF φ *k-degenerate* if there exists a variable ordering $x_1, \ldots, x_n$ in which $|\Delta^i| \leq k$ for all $i = 1, 2, \ldots, n$. We call a variable ordering $x_1, \ldots, x_n$ *smallest last* as in [51] if x_i is chosen in the order $i = n, n-1, \ldots, 1$, such that $|\Delta^i|$ is smallest for all variables that were not chosen. Clearly, a smallest last ordering gives the least k such that φ is k-degenerate. Therefore, we can check for every integer $k \geq 1$ whether φ is k-degenerate in $O(\|\varphi\|)$ time. If this holds, then we have $|\rho_{(t,i)}| \leq n^k$ and $T_{(t,i)} = O(kn^{k+1})$ for every $t \in PI(f)$ and $i \in V(t)$ (for $T_{(t,i)}$, apply the distributive law to $\Delta^i[t]$, and remove terms t, where some $x_j \in V(t)$ has no $c \in \Delta^i[t]$ such that $V(t) \cap V(c) = \{x_j\}$). Thus Theorem 3.4 implies the following.

THEOREM 4.1. *For k-degenerate CNFs φ,* DUALIZATION *is solvable with $O(\|\varphi\| \cdot n^{k+1})$ polynomial delay if $k \geq 1$ is constant.*

Applying the result of [36] that log-clause CNF is dualizable in incremental polynomial time, we obtain a polynomiality result also for nonconstant degeneracy.

THEOREM 4.2. *For $O(\log \|\varphi\|)$-degenerate CNFs φ, problem* DUALIZATION *is solvable in polynomial total time.*

In the following, we discuss several natural subclasses of degenerate CNFs.

4.1.1. Read-bounded CNFs. A monotone CNF φ is called *read-k* if each variable appears in φ at most k times. Clearly, read-k CNFs are k-degenerate, and in fact φ is read-k iff it is k-degenerate under every variable ordering. By applying Theorems 4.1 and 4.2, we obtain the following result.

COROLLARY 4.3. *For read-k CNFs φ, problem* DUALIZATION *is solvable*

(i) *with $O(\|\varphi\| \cdot n^{k+1})$ polynomial delay if k is constant and*

(ii) *in polynomial total time if $k = O(\log \|\varphi\|)$.*

Note that Corollary 4.3.(i) trivially implies that DUALIZATION is solvable in $O(|\psi| \cdot n^{k+2})$ time for constant k since $\|\varphi\| \leq kn$. This improves upon the previous best known algorithm [10], which is only $O(|\psi| \cdot n^{k+3})$ time, not polynomial delay, and outputs $PI(f)$ in no specific order. Corollary 4.3 (ii) is a nontrivial generalization of the result in [10], which was posed as an open problem [13].

4.1.2. Acyclic CNFs. Like in graphs, acyclicity is appealing in hypergraphs, respectively, monotone CNFs, from a theoretical as well as a practical point of view. However, there are many notions of acyclicity for hypergraphs (cf. [16]) since different generalizations from graphs are possible. We refer to α-, β-, γ-, and *Berge*-acyclicity as stated in [16], for which the following proper inclusion hierarchy is known:

$$\text{Berge-acyclic} \subseteq \gamma\text{-acyclic} \subseteq \beta\text{-acyclic} \subseteq \alpha\text{-acyclic}.$$

The notion of α-acyclicity came up in relational database theory. A monotone CNF φ is *α-acyclic* iff $\varphi = 1$ or is reducible by the Graham–Yu–Ozsoyoglu-reduction (GYO-reduction) [24, 52], i.e., repeated application of one of the following two rules to 0 (i.e., the empty clause):

(1) If variable x_i occurs in only one clause c, remove x_i from c.

(2) If distinct clauses c and c' satisfy $V(c) \subseteq V(c')$, remove c from φ.

Note that α-acyclicity of a monotone CNF φ can be checked, and a suitable GYO-reduction can be output, in $O(\|\varphi\|)$ time [49]. A monotone CNF φ is *β-acyclic* iff every CNF consisting of clauses in φ is α-acyclic. As shown in [15], the prime implicants of a monotone f represented by a β-acyclic CNF φ can be enumerated (and thus

DUALIZATION solved) in $p(\|\varphi\|) \cdot |\psi|$ time, where p is a polynomial in $\|\varphi\|$. However, the time complexity of DUALIZATION for the more general α-acyclic prime CNFs was left as an open problem. We now show that it is solvable with polynomial delay by showing that α-acyclic CNFs are 1-degenerate.

Let $\varphi \neq 1$ be a prime CNF. Let $a = a_1, a_2, \ldots, a_q$ be a GYO-reduction for φ, where $a_\ell = x_i$ if the ℓth operation removes x_i from c and where $a_\ell = c$ if it removes c from φ. Consider the unique variable ordering $b_1, b_2, \ldots, b_n$ such that b_i occurs after b_j in a for all $i < j$. For example, let $\varphi = c_1c_2c_3c_4$, where $c_1 = (x_1 \vee x_2 \vee x_3)$, $c_2 = (x_1 \vee x_3 \vee x_5)$, $c_3 = (x_1 \vee x_5 \vee x_6)$, and $c_4 = (x_3 \vee x_4 \vee x_5)$. Then φ is α-acyclic since it has the GYO-reduction

$$a_1 = x_2,\ a_2 = c_1,\ a_3 = x_4,\ a_4 = x_6,\ a_5 = c_4,\ a_6 = c_3,\ a_7 = x_1,\ a_8 = x_3,\ a_9 = x_5.$$

From this sequence, we obtain the variable ordering

$$b_1 = x_5,\ b_2 = x_3,\ b_3 = x_1,\ b_4 = x_6,\ b_5 = x_4,\ b_6 = x_2.$$

As easily checked, this ordering shows that φ is 1-degenerate. Under this ordering, we have $\Delta^1 = \Delta^2 = 1$, $\Delta^3 = (x_1 \vee x_3 \vee x_5)$, $\Delta^4 = (x_1 \vee x_5 \vee x_6)$, $\Delta^5 = (x_3 \vee x_4 \vee x_5)$, and $\Delta^6 = (x_1 \vee x_2 \vee x_3)$. This is not accidental.

LEMMA 4.4. *Every α-acyclic prime CNF is 1-degenerate.*

Note that the converse is not true; i.e., there exists a 1-degenerate CNF that is not α-acyclic. For example, $\varphi = (x_1 \vee x_2 \vee x_3)(x_1 \vee x_2 \vee x_4)(x_2 \vee x_3 \vee x_4 \vee x_5)$ is such a CNF. Lemma 4.4 and Theorem 4.1 imply the following result.

COROLLARY 4.5. *For α-acyclic CNFs φ, problem* DUALIZATION *is solvable with $O(\|\varphi\| \cdot n^2)$ delay.*

Observe that, for a prime α-acyclic φ, we have $|\varphi| \leq n$. Thus, if we slightly modify Algorithm DUALIZE to check $\Delta^i = 1$ in advance (which can be done in linear time in a preprocessing phase) such that such Δ^i need not be considered in Step 2, then the resulting algorithm has $O(n \cdot |\varphi| \cdot \|\varphi\|)$ delay. Observe that the algorithm in [15] solves, minorly adapted for enumerative output, DUALIZATION for β-acyclic CNFs with $O(n \cdot |\varphi| \cdot \|\varphi\|)$ delay. Thus the above modification of DUALIZE is of the same order.

4.1.3. CNFs with bounded treewidth. A *tree decomposition (of type* I*)* of a monotone CNF φ is a tree $T = (W, E)$, where each node $w \in W$ is labeled with a set $X(w) \subseteq V(\varphi)$ under the following conditions:

1. $\bigcup_{w \in W} X(w) = V(\varphi)$;
2. for every clause c in φ, there exists some $w \in W$ such that $V(c) \subseteq X(w)$; and
3. for any variable $x_i \in V$, the set of nodes $\{w \in W \mid x_i \in X(w)\}$ induces a (connected) subtree of T.

The *width* of T is $\max_{w \in W} |X(w)| - 1$, and the *treewidth* of φ, denoted by $Tw_1(\varphi)$, is the minimum width over all its tree decompositions.

Note that the usual definition of treewidth for a graph [46] results in the case in which φ is a 2-CNF. Similarly to acyclicity, there are several notions of treewidth for hypergraphs, respectively, monotone CNFs. For example, tree decomposition of type II of CNF $\varphi = \bigwedge_{c \in C} c$ is defined as type-I tree decomposition of its incident 2-CNF (i.e., graph) $G(\varphi)$ [11, 23]. That is, for each clause $c \in \varphi$, we introduce a new variable y_c and construct $G(\varphi) = \bigwedge_{x_i \in c \in \varphi}(x_i \vee y_c)$. (Here, $x_i \in c$ denotes that x_i appears in c.) Let $Tw_2(\varphi)$ denote the type-II treewidth of φ.

PROPOSITION 4.6. *For every monotone CNF φ, it holds that $Tw_2(\varphi) \leq Tw_1(\varphi) + 2^{Tw_1(\varphi)+1}$.*

Proof. Let $T = (W, E)$, $X : W \to 2^V$ be any tree decomposition of φ having width $Tw_1(\varphi)$. Introduce for all $c \in \varphi$ new variables y_c, and add y_c to every $X(w)$ such that $V(c) \subseteq X(w)$. Clearly, the result is a type-I tree decomposition of $G(\varphi)$ and thus a type-II tree decomposition of φ. Since at most $2^{|X(w)|}$ many y_c are added to $X(w)$ and $|X(w)| - 1 \leq Tw_1(\varphi)$ for every $w \in W$, the result follows. □

This means that if $Tw_1(\varphi)$ is bounded by some constant, then so is $Tw_2(\varphi)$. Moreover, $Tw_1(\varphi) = k$ implies that φ is a k-CNF; we discuss k-CNFs in section 4.2 and consider only $Tw_2(\varphi)$ here. The following proposition states some relationships between type-II treewidth and other restrictions of CNFs from above.

PROPOSITION 4.7. *The following properties hold for type-*II *treewidth.*

(i) *There is a family of monotone prime CNFs φ such that $Tw_2(\varphi)$ is bounded by a constant, but φ is not k-CNF for any constant k.*

(ii) *There is a family of monotone prime CNFs φ such that $Tw_2(\varphi)$ is bounded by a constant, but φ does not have bounded read.*

(iii) *There is a family of α-acyclic prime CNFs φ such that $Tw_2(\varphi)$ is not bounded by any constant. (This is a contrast to the graph case that a graph is acyclic if and only if its treewidth is* 1*.)*

Proof. (i) For example, $\varphi = (\bigvee_{x_i \in V} x_i)$ has $Tw_2(\varphi) = 1$, since it has a tree decomposition $T = (W, E)$ with $X : W \to 2^V$ defined by $W = \{1, 2, \ldots, n\}$, $E = \{(w, w+1),\ w = 1, 2, \ldots, n-1\}$, and $X(w) = \{x_w, y_c\}$, $w \in W$, where $c = (\bigvee_{x_i \in V} x_i)$. However, it is not an $(n-1)$-CNF (but an n-CNF). On the other hand, by Lemma 4.8, we can see that there is a family of monotone prime CNFs φ such that $Tw_2(\varphi)$ is not bounded by any constant, but φ is k-CNF for some constant k.

(ii) For example, let φ be a CNF containing $n-1$ clauses $c_i = (x_1 \vee x_i)$, $i = 2, 3, \ldots, n$. Then φ has $Tw_2(\varphi) = 1$, since it has a tree decomposition $T = (W, E)$ with $X : W \to 2^V$ defined by $W = \{(c_i, x_1), (c_i, x_i),\ i = 2, 3, \ldots, n\}$, $E = \{((c_i, x_1), (c_{i+1}, x_1)),\ i = 2, 3, \ldots, n-1\} \cup \{((c_i, x_1), (c_i, x_i)),\ i = 2, 3, \ldots, n\}$, and $X((c_i, x_k)) = \{y_{c_i}, x_k\}$, $(c_i, x_k) \in W$. However, it is not read-$(n-2)$ (but read-$(n-1)$).

(iii) For example, let φ be a CNF on $V = \{x_1, x_2, \ldots, x_{2n}\}$ containing n clauses $c_i = (x_i \vee \bigvee_{j \geq n+1} x_j)$ for $i = 1, \ldots, n$. Then φ is α-acyclic. We claim that $Tw_2(\varphi) \geq n-1$. Let us assume that there exists a tree $T = (W, E)$ with $X : E \to 2^V$ that shows $Tw_2(\varphi) \leq n-2$, where T is regarded as a rooted tree. Let $T_i = (W_i, E_i)$ be the subtree of T induced by $W_i = \{w \in W \mid y_{c_i} \in X(w)\}$, and let r_i be its root. Consider the case in which W_i and W_j are disjoint for some i and j. Suppose that r_j is an ancestor of r_i. Since $|X(r_i)| \leq Tw_2(\varphi) + 1 \leq n-1$, there exists a node $x_{n+k} \in V$ such that $1 \leq k \leq n$ and $x_{n+k} \notin X(r_i)$. However, since the incident graph of φ contains two edges (x_{n+k}, y_{c_i}) and (x_{n+k}, y_{c_j}), we have $x_{n+k} \in \bigcup_{w \in W_i - \{r_i\}} X(w)$ and $x_{n+k} \in \bigcup_{w \in W_j} X(w)$. This is a contradiction to the condition that $\{w \in W \mid x_{n+k} \in X(w)\}$ is connected. Similarly, we can prove our claim when T_i and T_j are disjoint, but r_j is not an ancestor of r_i.

We thus consider the case in which $W_i \cap W_j \neq \emptyset$ holds for any i and j. Since the T_i's are trees, the family of W_i, $i = 1, 2, \ldots, n$, satisfies the well-known Helly property; i.e., there exists a node w in $\bigcap_{i=1}^{n} W_i$. $X(w)$ must contain all y_{c_i}'s. This implies $|X(w)| \geq n$, which is a contradiction. □

As we show now, bounded treewidth implies bounded degeneracy.

LEMMA 4.8. *Let φ be any monotone CNF with $Tw_2(\varphi) = k$. Then φ is 2^k-degenerate.*

Proof. Let $T = (W, E)$ with $X : W \to 2^V$ show $Tw_2(\varphi) = k$. From this, we reversely construct a variable ordering $a = a_1, \ldots, a_n$ on $V = V(\varphi)$ such that

$|\Delta^i| \leq 2^k$ for all i.

Set $i := n$. Choose any leaf w^* of T, and let $p(w^*)$ be a node in W adjacent to w^*. If $X(w^*) \setminus X(p(w^*)) \subseteq \{y_c \mid c \in \varphi\}$, then remove w^* from T. On the other hand, if $(X(w^*) \setminus X(p(w^*))) \cap V = \{x_{j_1}, \ldots, x_{j_\ell}\}$, where $\ell \geq 1$ (in this case, only $X(w^*)$ contains $x_{j_1}, \ldots, x_{j_\ell}$), then define $a_{i+1-h} = x_{j_h}$ for $h = 1, \ldots, \ell$, and update $i := n - \ell$, $X(w^*) := X(w^*) \setminus \{x_{j_1}, \ldots, x_{j_\ell}\}$, and $X(w) := X(w) \setminus \{y_c \mid c \in \varphi, V(c) \cap \{x_{j_1}, \ldots, x_{j_\ell}\} \neq \emptyset\}$ for every $w \in W$. Let a be completed by repeating this process.

We claim that a shows that $|\Delta^i| \leq 2^k$ for all $i = 1, \ldots, n$. To see this, let w^* be chosen during this process, and assume that $a_i \in X(w^*) \setminus X(p(w^*))$. Then, by induction on the (reverse) construction of a, we obtain that, for each clause $c \in \Delta^i$, we must have either (a) $y_c \in X(w^*)$ or (b) $V(c) \subseteq X(w^*)$. The latter case may arise if in previous steps of the process some descendant $d(w^*)$ of w^* was removed which contains y_c such that y_c does not occur in w^*; however, in this case $V(c) \subseteq X(w)$ must be true on every node on the path from $d(w^*)$ to w^*.

Now let $q = |X(w^*) \setminus V|$. Since $|X(w^*) \setminus \{a_i\}| \leq k$, we have

$$|\Delta^i| \;\leq\; q + 2^{k-q} \;\leq\; 2^k.$$

This proves the claim. □

COROLLARY 4.9. *For CNFs φ with $Tw_2(\varphi) \leq k$,* DUALIZATION *is solvable* (i) *with $O(\|\varphi\| \cdot n^{2^k+1})$ polynomial delay if k is constant and* (ii) *in polynomial total time if $k = O(\log\log\|\varphi\|)$.*

4.2. Recursive application of Algorithm DUALIZE. Algorithm DUALIZE computes in Step 2 the prime DNF $\rho_{(t,i)}$ of the function represented by $\Delta^i[t]$. Since $\Delta[t]$ is the prime CNF of some monotone function, we can recursively apply DUALIZE to $\Delta^i[t]$ for computing $\rho_{(t,i)}$. Let us call this variant R-DUALIZE. Then we have the following result.

THEOREM 4.10. *If its recursion depth is d,* R-DUALIZE *solves* DUALIZATION *in $O(n^{d-1} \cdot |\psi|^{d-1} \cdot \|\varphi\|)$ time.*

Proof. If $d = 1$, then $\Delta^i[t_{min}] = 1$ holds for t_{min} and every $i \geq 1$. This means that $PI(f) = \{t_{min}\}$ and φ is a 1-CNF (i.e., each clause in φ contains exactly one variable). Thus, in this case, R-DUALIZE needs $O(n)$ time. Recall that Algorithm DUALIZE needs, by (6), time $\sum_{t \in PI(f)} \sum_{x_i \in V(t)} (T_{(t,i)} + |\rho_{(t,i)}| \cdot O(\|\varphi\|))$. If $d = 2$, then $T_{(t,i)} = O(n)$ and $|\rho_{(t,i)}| \leq 1$. Therefore, R-DUALIZE needs time $O(n \cdot |\psi| \cdot \|\varphi\|)$. For $d \geq 3$, Corollary 3.5.(ii) implies that R-DUALIZE needs $O(n^{d-1} \cdot |\psi|^{d-1} \cdot \|\varphi\|)$ time. □

Recall that a CNF φ is called *k-CNF* if each clause in φ has at most k literals. Clearly, if we apply Algorithm R-DUALIZE to a monotone k-CNF φ, the recursion depth of R-DUALIZE is at most k. Thus we obtain the following result; it re-establishes, with different means, the main positive result of [6, 15].

COROLLARY 4.11. R-DUALIZE *solves* DUALIZATION *in $O(n^{k-1} \cdot |\psi|^{k-1} \cdot \|\varphi\|)$ time, i.e., in polynomial total time for monotone k-CNFs φ where k is constant.*

5. Limited nondeterminism. In the previous section, we discussed polynomial cases of monotone dualization. In this section, we now turn to the issue of the precise complexity of this problem. For this purpose, we consider the decision problem DUAL, i.e., decide whether given monotone prime CNFs φ and ψ represent dual Boolean functions instead of the search problem DUALIZATION.

It appears that problem DUAL can be solved with limited nondeterminism, i.e., with polylog many guessed bits by a polynomial time nondeterministic Turing machine. This result might bring new insight toward settling the complexity of the problem.

We adopt Kintala and Fischer's terminology [33] and write $g(n)$-P for the class of sets accepted by a nondeterministic Turing machine in polynomial time making at most $g(n)$ nondeterministic steps on every input of length n. For every integer $k \geq 1$, define $\beta_k\mathrm{P} = \bigcup_c (c \log^k n)$-P. The $\beta\mathrm{P}$ *Hierarchy* consists of the classes

$$\mathrm{P} = \beta_1\mathrm{P} \subseteq \beta_2\mathrm{P} \subseteq \cdots \subseteq \bigcup_k \beta_k\mathrm{P} = \beta\mathrm{P}$$

and lies between P and NP. The β_kP classes appear to be rather robust; they are closed under polynomial time and logspace many-one reductions and have complete problems (cf. [22]). The complement class of β_kP is denoted by co-β_kP.

We start in section 5.1 by recalling Algorithm A of [17], reformulated for CNFs and by analyzing A's behavior. The proof that A can be converted to an algorithm that uses $\log^3 n$ nondeterministic bit guesses, and that DUAL is thus in co-β_3P, is rather easy and should give the reader a sense of how our new method of analysis works. In section 5.2, we use basically the same technique for analyzing the more involved Algorithm B of [17]. Using a modification of this algorithm, we show that DUAL is in co-β_2P. We also prove the stronger result that the complement of DUAL can be solved in polynomial time with only $O(\chi(n) \cdot \log(n))$ nondeterministic steps (= bit guesses). Finally, section 5.3 shows that membership in co-β_2P can alternatively be obtained by combining the results of [17] with a theorem of Beigel and Fu [2].

5.1. Analysis of Algorithm A of Fredman and Khachiyan. The first algorithm in [17] for recognizing dual monotone pairs is as follows.

Algorithm A (reformulated for CNFs[1]).

Input: Monotone CNFs φ, ψ representing monotone f, g s.t. $V(c) \cap V(c') \neq \emptyset$ for all $c \in \varphi$, $c' \in \psi$.

Output: `yes` if $f = g^d$, and otherwise a vector w of form $w = (w_1, \ldots, w_m)$ such that $f(w) \neq g^d(w)$.

Step 1:

Delete all redundant (i.e., nonminimal) clauses from φ and ψ.

Step 2:

Check that (1) $V(\phi) = V(\psi)$, (2) $\max_{c \in \varphi} |c| \leq |\psi|$, (3) $\max_{c' \in \psi} |c'| \leq |\varphi|$, and (4) $\Sigma_{c \in \varphi} 2^{-|c|} + \Sigma_{c' \in \psi} 2^{-|c'|} \geq 1$. If any of conditions (1)–(4) fails, $f \neq g^d$ and a witness w is found in polynomial time (cf. [17]).

Step 3:

If $|\varphi| \cdot |\psi| \leq 1$, test duality in $O(1)$ time.

Step 4:

If $|\varphi| \cdot |\psi| \geq 2$, find the lowest index variable x_i which occurs in φ or ψ with frequency $\geq 1/\log(|\varphi| + |\psi|)$.

Let

$$\varphi_0 = \{c - \{x_i\} \mid x_i \in c,\, c \in \varphi\}, \quad \varphi_1 = \{c \mid x_i \notin c,\, c \in \varphi\},$$

[1] In [17], duality is tested for DNFs, while our problem DUAL speaks about CNFs; this is insignificant since DNFs are trivially translated to CNFs for this task and vice versa (cf. section 2).

$\psi_0 = \{c' - \{x_i\} \mid x_i \in c',\ c' \in \psi\},\ \psi_1 = \{c' \mid x_i \notin c',\ c' \in \psi\}.$

Call Algorithm A on the two pairs of forms

(A.1) $(\varphi_1, \psi_0 \wedge \psi_1)$ and (A.2) $(\psi_1, \varphi_0 \wedge \varphi_1)$.

If both calls return `yes`, then return `yes` (as $f = g^d$); otherwise, we obtain w such that $f(w) \neq g^d(w)$ in polynomial time (cf. [17]).

We observe that, as noted in [17], the binary length of any standard encoding of the input φ, ψ to Algorithm A is polynomially related to $|\varphi| + |\psi|$ if Step 3 is reached. Thus, for our purpose, we consider $|\varphi| + |\psi|$ to be the input size.

Let φ^*, ψ^* be the original input for A. For any pair (φ, ψ) of CNFs, define its *volume* by $v = |\varphi| \cdot |\psi|$, and let $\epsilon = 1/\log n$, where $n = |\varphi^*| + |\psi^*|$. As shown in [17], Step 4 of Algorithm A divides the current (sub)problem of volume $v = |\varphi| \cdot |\psi|$ by self-reduction into subproblems (A.1) and (A.2) of respective volumes, assuming without loss of generality that x_i frequently occurs in φ (otherwise, swap φ and ψ):

$$|\varphi_1| \cdot |\psi_0 \wedge \psi_1| \leq (1 - \epsilon) \cdot v, \tag{7}$$

$$|\varphi_0 \wedge \varphi_1| \cdot |\psi_1| \leq |\varphi| \cdot (|\psi| - 1) \leq v - 1. \tag{8}$$

Let $T = T(\varphi, \psi)$ be the recursion tree generated by A on input (φ, ψ). In T, each node u is labeled with the respective monotone pair, denoted by $I(u)$; thus, if r is the root of T, then $I(r) = (\varphi, \psi)$. The *volume* $v(u)$ of node u is defined as the volume of its label $I(u)$.

Any node u is a leaf of T if Algorithm A stops on input $I(u) = (\varphi, \psi)$ during Steps 1–3; otherwise, u has a left child u_l and a right child u_r corresponding to (A.1) and (A.2), i.e., labeled $(\varphi_1, \psi_0 \wedge \psi_1)$ and $(\psi_1, \varphi_0 \wedge \varphi_1)$, respectively. Without loss of generality, u_l is the "high frequency move" by the splitting variable.

We observe that every node u in T is determined by a *unique path* from the root to u in T and thus by a unique sequence $seq(u)$ of right and left moves starting from the root of T and ending at u. The following key lemma bounds the number of moves of each type for certain inputs.

LEMMA 5.1. *Suppose* $|\varphi^*| + |\psi^*| \leq |\varphi^*| \cdot |\psi^*|$. *Then for any node* a *in* T, $seq(a)$ *contains at most* v^* *right moves and at most* $\log^2 v^*$ *left moves, where* $v^* = |\varphi^*| \cdot |\psi^*|$.

Proof. By (7) and (8), each move decreases the volume of a node label. Thus the length of $seq(u)$, and in particular the number of right moves, is bounded by v^*. To obtain the better bound for the left moves, we will use the following well-known inequality:

$$(1 - 1/y)^y \leq 1/e \qquad \text{for } y \geq 1. \tag{9}$$

In fact, the sequence $(1 - 1/y_i)^{y_i}$ for any $1 \leq y_1 < y_2 < \cdots$ monotonically converges to $1/e$ from below. By (7), the volume $v(u)$ of any node u such that $seq(u)$ contains $\log^2 v^*$ left moves is bounded as follows:

$$v(u) \leq v^* \cdot (1 - \epsilon)^{\log^2 v^*} = v^* \cdot (1 - 1/\log n)^{\log^2 v^*}.$$

Since $n = |\varphi^*| + |\psi^*| \leq |\varphi^*| \cdot |\psi^*| = v^*$ and because of (9), it follows that

$$\begin{aligned} v(u) &\leq v^* \cdot \left((1 - 1/\log v^*)^{\log v^*}\right)^{\log v^*} \\ &\leq v^* \cdot (1/e)^{\log v^*} = v^*/(e^{\log v^*}) < v^*/(2^{\log v^*}) = 1. \end{aligned}$$

Thus u must be a leaf in T. Hence, for every u in T, $seq(u)$ contains at most $\log^2 v^*$ left moves. □

THEOREM 5.2. *Problem* DUAL *is in* co-β_3P.

Proof. Instances such that either $c \cap c' = \emptyset$ for some $c \in \varphi^*$ and $c' \in \psi^*$, the sequence $seq(u)$ is empty, or $|\varphi^*| + |\psi^*| > |\varphi^*| \cdot |\psi^*|$ are easily recognized and solved in deterministic polynomial time. In the remaining cases, if $f \neq g^d$, then there exists a leaf u in T labeled by a nondual pair (φ', ψ'). If $seq(u)$ is known, we can compute, by simulating Algorithm A on the branch described by $seq(u)$, the entire path $u_0, u_1, \ldots, u_l = u$ from the root u_0 to u with all labels $I(u_0) = (\varphi^*, \psi^*)$, $I(u_1), \ldots, I(u_l)$ and check that $I(u_l)$ is nondual in Steps 2 and 3 of Algorithm A in polynomial time. Since the binary length of any standard encoding of (φ^*, ψ^*) is polynomially related to $n = |\varphi^*| + |\psi^*|$ if $seq(u)$ is nonempty, to prove the result, it is sufficient to show that $seq(u)$ can be constructed in polynomial time from $O(\log^3 v^*)$ suitably guessed bits. To see this, let us represent every $seq(u)$ as a sequence $seq^*(u) = [\ell_0, \ell_1, \ell_2, \ldots, \ell_k]$, where ℓ_0 is the number of leading right moves and ℓ_i is the number of consecutive right moves after the ith left move in $seq(u)$ for $i = 1, \ldots, k$. For example, if $seq(u) = [\mathtt{r}, \mathtt{r}, \mathtt{l}, \mathtt{r}, \mathtt{r}, \mathtt{r}, \mathtt{l}]$, then $seq^*(u) = [2, 3, 0]$. By Lemma 5.1, $seq^*(u)$ has length at most $\log^2 v^* + 1$. Thus $seq^*(u)$ occupies in binary only $O(\log^3 v)$ bits; moreover, $seq(u)$ is trivially computed from $seq^*(u)$ in polynomial time. □

5.2. Analysis of Algorithm B of Fredman and Khachiyan. The aim of the above proof was to exhibit a new method of algorithm analysis that allows us to show with very simple means that duality can be polynomially checked with limited nondeterminism. By applying the same method of analysis to the slightly more involved Algorithm B of [17] (which runs in $n^{4\chi(n)+O(1)}$ time, and thus in $n^{o(\log n)}$ time), we can sharpen the above result by proving that deciding whether monotone CNFs φ and ψ are nondual is feasible in polynomial time with $O(\chi(n) \cdot \log n)$ nondeterministic steps; consequently, the problem DUAL is in co-β_2P.

Like Algorithm A, Algorithm B uses a recursive self-reduction method that decomposes its input, a pair (φ, ψ) of monotone CNFs, into smaller input instances for recursive calls. Analogously, the algorithm is thus best described via its *recursion tree* T, whose root represents the input instance (φ^*, ψ^*) (of size n), whose intermediate nodes represent smaller instances, and whose leaves represent those instances that can be solved in polynomial time. Like for Algorithm A, the nodes u in T are labeled with the respective instances $I(u) = (\varphi, \psi)$ of monotone pairs. Whenever there is a branching from a node u to children, then $I(u)$ is a pair of dual monotone CNFs iff $I(u')$ for *each* child u' of u in T is a pair of dual monotone CNFs. Therefore, the original input (φ^*, ψ^*) is a dual monotone pair iff all leaves of T are labeled with dual monotone pairs.

Rather than describing Algorithm B in full detail, we confine ourselves here to recalling those features which are relevant for our analysis. In particular, we will describe some essential features of its recursion tree T.

For each variable x_i occurring in φ, the *frequency* ϵ_i^φ *of* x_i *with respect to* φ is defined as $\epsilon_i^\varphi = \frac{|\{c \in \varphi : x_i \in c\}|}{|\varphi|}$, i.e., as the number of clauses of φ containing x_i divided by the total number of clauses in φ. Moreover, for each $v \geq 1$, let $\chi(v)$ be defined by $\chi(v)^{\chi(v)} = v$.

Let $v^* = |\varphi^*||\psi^*|$ denote the volume of the input (= root) instance (φ^*, ψ^*). For the rest of this section, we assume that $|\varphi^*| + |\psi^*| \leq |\varphi^*| \cdot |\psi^*|$. In fact, in any instance which violates this inequality, either φ^* or ψ^* has at most one clause; in this case,

DUAL is trivially solvable in polynomial time.

Algorithm B first constructs the root r of T and then recursively expands the nodes of T. For each node u with label $I(u) = (\varphi, \psi)$, Algorithm B does the following.

The algorithm first performs a polynomial time computation, which we shall refer to as LCHECK(φ, ψ) here, as follows. LCHECK(φ, ψ) first eliminates all redundant (i.e., nonminimal) clauses from φ and ψ and then tests whether at least one of the following conditions is violated:

1. $V(\varphi) = V(\psi)$;
2. $\max_{c \in \varphi} |c| \leq |\psi|$ and $\max_{c \in \psi} |c| \leq |\varphi|$;
3. $\min(\, |\varphi|, |\psi|\,) > 2$.

If LCHECK$(\varphi, \psi) = \mathit{true}$, then u is a leaf of T (i.e., not further expanded); whether $I(\varphi, \psi)$ is a dual monotone pair is then decided by some procedure TEST(φ, ψ) in polynomial time. In case TEST(φ, ψ) returns *false*, the original input (φ^*, ψ^*) is not a dual monotone pair, and Algorithm B returns *false*. Moreover, in this case a counterexample w to the duality of φ^* and ψ^* is computable in polynomial time from the path leading from the root r of T to u.

If LCHECK(φ, ψ) returns *false*, Algorithm B chooses in polynomial time some appropriate variable x_i such that $\epsilon_i^{\varphi} > 0$ and $\epsilon_i^{\psi} > 0$ and creates two or more children of u by deterministically choosing one of three alternative decomposition rules, (i), (ii), or (iii). Each rule decomposes $I(u) = (\varphi, \psi)$ into smaller instances, whose respective volumes are summarized as follows. Let, as for Algorithm A, $\varphi_0 = \{c - \{x_i\} \mid x_i \in c,\, c \in \varphi\}$, $\varphi_1 = \{c \mid x_i \notin c,\, c \in \varphi\}$, $\psi_0 = \{c' - \{x_i\} \mid x_i \in c',\, c' \in \psi\}$, and $\psi_1 = \{c' \mid x_i \notin c',\, c' \in \psi\}$. Furthermore, define $\epsilon(v) = 1/\chi(v)$ for any $v > 0$.

Rule (i) If $\epsilon_i^{\varphi} \leq \epsilon(v(u))$, then $I(u)$ is decomposed into:

- (a) one instance $(\varphi_1, \psi_0 \wedge \psi_1)$ of volume $\leq (1 - \epsilon_i^{\varphi}) \cdot v(u)$;
- (b) $|\psi_0|$ instances $I_1, \ldots, I_{|\psi_0|}$ of volume $\leq \epsilon_i^{\varphi} \cdot v(u)$ each. Each such instance I_j corresponds to one clause of ψ_0 and can thus be identified as the jth clause of ψ_0 with an index $j \leq |\psi_0| < n$ (recall that n denotes the size of the original input).

Rule (ii) If $\epsilon_i^{\varphi} > \epsilon(v(u)) \geq \epsilon_i^{\psi}$, then $I(u)$ is decomposed into:

- (a) one instance $(\psi_1, \varphi_0 \wedge \varphi_1)$ of volume $\leq (1 - \epsilon_i^{\psi}) \cdot v(u)$;
- (b) $|\varphi_0|$ instances $I_1, \ldots, I_{|\varphi_0|}$ of volume $\leq \epsilon_i^{\psi} \cdot v(u)$ each. Each such instance I_j corresponds to one clause of φ_0 and can be identified by an index $j \leq |\varphi_0| < v^*$.

Rule (iii) If both $\epsilon_i^{\varphi} > \epsilon(v(u))$ and $\epsilon_i^{\psi} > \epsilon(v(u))$, then I is decomposed into:

- (c_0) one instance of volume $\leq (1 - \epsilon_i^{\varphi}) \cdot v(u)$, and
- (c_1) one instance of volume $\leq (1 - \epsilon_i^{\psi}) \cdot v(u)$.

Algorithm B returns *true* iff TEST$(I(u))$ returns *true* for each leaf u of the recursion tree. This concludes the description of Algorithm B.

For each node u and child u' of u in T, we label the arc (u, u') with the precise type of rule that was used to generate u' from u. The possible labels are thus (i.a), (i.b), (ii.a), (ii.b), (iii.c_0), and (iii.c_1). We call (i.a) and (ii.a) *a-labels*, (i.b) and (ii.b) *b-labels*, and (iii.c_0) and (iii.c_1) *c-labels*. Any arc with a *b*-label is in addition labeled with the index j of the respective instance I_j in the decomposition, which we refer to as the *j-label* of the arc.

DEFINITION 5.1. *For any node u of the tree T, let seq(u) denote the sequence of all edge-labels on the path from the root r of T to u.*

Clearly, if *seq*(u) is known, then the entire path from r to u including all node-labels (in particular, the one of u) can be computed in polynomial time. Indeed,

the depth of the tree is at most v^*, and adding a child to a node of T according to Algorithm B is feasible in polynomial time.

The following lemma bounds the number of various labels which may occur in $seq(u)$.

LEMMA 5.3. *For each node u in T, $seq(u)$ contains at most*

(i) v^* *many a-labels,*
(ii) $\log v^*$ *many b-labels, and*
(iii) $\log^2 v^*$ *many c-labels.*

Proof. (i) Let us consider rule (i.a) first. Given that $\epsilon_i^{\varphi} > 0$, x_i effectively occurs in some clause of φ. Thus $|\varphi_1| < |\varphi|$. Moreover, by definition of ψ_0 and ψ_1, $|\psi_0 \wedge \psi_1| \leq |\psi|$. Thus we have $|\varphi_1| \cdot |\psi_0 \wedge \psi_1| < |\varphi| \cdot |\psi|$. It follows that whenever rule (i.a) is applied, the volume decreases (at least by 1). The same holds for rule (ii.a) by a symmetric argument. Since no rule ever increases the volume, there are at most v^* applications of an a-rule.

(ii) Assume that rule (i.b) is applied to generate a child t' of node t. By condition 3 of LCHECK, $v(t) > 4$. Therefore, $\chi(v(t)) > 2$, and thus $\epsilon_i^{\varphi} \leq \epsilon(v(t)) < 1/2$. It follows that $v(t') < v(t)/2$. The same holds if t' results from t via rule (ii.b). Because no rule ever increases the volume, any node generated after (among others) $\log v^*$ applications of a b-rule has volume ≤ 1 and is thus a leaf in T.

(iii) If a c-rule is applied to generate a child t' of a node t, and since $\epsilon(v(t)) > \epsilon(v^*) > 1/\log v^*$, the volume of $v(t)$ decreases at least by factor $(1 - 1/\log v^*)$. Thus the volume of any node u which results from t after $\log v^*$ applications of a c-rule satisfies $v(u) \leq v(t)(1 - 1/\log v^*)^{\log v^*} \leq v(t)/e$ by (9); i.e., the volume has decreased by more than half. Thus any node u resulting from the root of T after $\log^2 v^*$ applications of a c-rule satisfies $v(u) \leq v^* \cdot (\frac{1}{2})^{\log v^*} = 1$; that is, u is a leaf in T. □

THEOREM 5.4. *Deciding whether monotone CNFs φ and ψ are nondual is feasible in polynomial time with $O(\log^2 n)$ nondeterministic steps, where $n = |\varphi| + |\psi|$.*

Proof. As in the proof of Theorem 5.2, we use a compact representation $seq^*(u)$ of $seq(u)$. However, here the definition of seq^* is somewhat more involved.

1. $seq^*(u)$ contains all b-labels of $seq(u)$, which are the anchor elements of $seq^*(u)$. Every b-label is immediately followed by its associated j-label, i.e., the label specifying which of the (many) b-children is chosen. We call a b-label and its associated j-label a *bj-block*.

2. At the beginning of $seq^*(u)$, as well as after each bj-block, there is an *ac-block*. The first ac-block in $seq^*(u)$ represents the sequence of all a- and c-labels in $seq(u)$ preceding the first b-label in $seq(u)$, and the ith ac-block in $seq^*(u)$, $i > 1$, represents the sequence of the a- and c-labels (uninterrupted by any other label) following the $(i-1)$st bj-block in $seq(u)$.

Each ac-block consists of an α-block followed by a γ-block, where

(i) the α-block contains, in binary, the *number* of a-labels in the ac-block, and
(ii) the γ-block contains all c-labels (single bits) in the ac-block, in the order they appear.

For example, if $s =$ "$(i.a), (ii.a), c_0, (ii.a), c_1, c_0, (i.a)$" is a maximal ac-subsequence in $seq(u)$, then its corresponding ac-block in $seq^*(u)$ is "$10, c_0, c_1, c_0$," where $10\,(= 4)$ is the α-block (stating that there are four a-labels) and "c_0, c_1, c_0" is the γ-block enumerating the c-labels in s in their correct order.

The following facts are now the key to the result.

FACT A. *Given ϕ^*, ψ^*, and a string s, it is possible to compute in polynomial time the path $r = u_0, u_1, \ldots, u_l = u$ from the root r of T to the unique node u in T*

such that $s = seq^*(u)$ and all labels $I(u_i)$, or to tell that no such node u exists (i.e., $s \neq seq^*(u)$ for every node u in T).

This can be done by a simple procedure, which incrementally constructs u_0, u_1, etc. as follows.

Create the root node $r = u_0$, and set $I(u_0) = (\phi^*, \psi^*)$ and $t := 0$. Generate the next node u_{t+1} and label it, while processing the main blocks (*ac*-blocks and *bj*-blocks) in s in order as follows.

ac-block. Suppose the α-block of the current *ac*-block has value n_α and the γ-block contains labels $\gamma_1, \ldots, \gamma_k$. Set up counters $p := 0$ and $q := 0$, and while $p < n_\alpha$ or $q < k$, do the following.

If LCHECK$(I(u_t)) = true$, then flag an error and halt, as $s \neq seq^*(u)$ for every node u in T. Otherwise, determine the rule type $\tau \in \{(\mathrm{i}), (\mathrm{ii}), (\mathrm{iii})\}$ used by Algorithm B to (deterministically) decompose $I(u_t)$.

1. If $\tau \in \{(\mathrm{i}), (\mathrm{ii})\}$ and $p < n_\alpha$, then assign $I(u_{t+1})$ the a-child of $I(u_t)$ according to Algorithm B, and increment p and t by 1.
2. If $\tau = (\mathrm{iii})$ and $q < k$, then increment q by 1, assign $I(u_{t+1})$ the γ_q-child of $I(u_t)$ according to Algorithm B, and increment t by 1.
3. In all other cases (i.e., either $\tau \in \{(\mathrm{i}), (\mathrm{ii})\}$ and $p \geq n_\alpha$ or $\tau = (\mathrm{iii})$ and $q \geq k$), flag an error and halt, since $s \neq seq^*(u)$ for every node u in T.

bj-block. Determine the rule type $\tau \in \{(\mathrm{i}), (\mathrm{ii}), (\mathrm{iii})\}$ used by Algorithm B to (deterministically) decompose $I(u_t)$. If $\tau = (\mathrm{iii})$, then flag an error and halt since $s \neq seq^*(u)$ for every node u in T. Otherwise, assign $I(u_{t+1})$ the j'th $(\tau.\mathbf{b})$-child of $I(u_t)$ according to rule $(\tau.\mathbf{b})$ of Algorithm B, where j' is the j-label of the current *bj*-block.

Clearly, this procedure outputs in polynomial time the desired labeled path from r to u or flags an error if $s \neq seq^*(u)$ for every node u in T.

Let us now bound the size of $seq^*(u)$ in terms of the original input size v^*.

FACT B. *For any u in T, the size of $seq^*(u)$ is $O(\log^2 v^*)$.*

By Lemma 5.3 (ii), there are $< \log v^*$ *bj*-blocks. As already noted, each *bj*-block has size $O(\log v^*)$; thus the total size of all *bj*-blocks is $O(\log^2 v^*)$. Next, there are at most $\log v^*$ many *ac*-blocks and thus α-blocks. Each α-block encodes a number of $< v^*$ a-rule applications (see Lemma 5.3.(i)) and thus uses at most $\log v^*$ bits. The total size of all α-blocks is thus at most $\log^2 v^*$. Finally, by Lemma 5.3 (iii), the total size of all γ-blocks is at most $\log^2 v^*$. Overall, this means that $seq^*(u)$ has size $O(\log^2 v^*)$.

To prove that Algorithm B rejects input (φ^*, ψ^*), it is thus sufficient to guess $seq^*(u)$ for some leaf u in T, to compute in polynomial time the corresponding path $r = u_0, u_1, \ldots, u_l = u$, and to verify that LCHECK$(I(u)) = true$ but TEST$(I(u)) = false$. Therefore, nonduality of ϕ^* and ψ^* can be decided in polynomial time with $O(\log^2 v^*)$ bit guesses. Given that $v^* \leq n^2$, the number of guesses is $O(\log^2 n^2) = O(\log^2 n)$. □

The following result is an immediate consequence of this theorem.

COROLLARY 5.5. *Problem* DUAL *is in* co-β_2P *and solvable in deterministic $n^{O(\log n)}$ time, where $n = |\varphi| + |\psi|$.*

(Note that yes-instances of DUAL must have size polynomial in n since dual monotone pairs (φ, ψ) must satisfy conditions (2) and (3) in Step 2 of Algorithm A.) We remark that the proofs of Lemma 5.3 and Theorem 5.4 did not stress the fact that $\epsilon(v) = 1/\chi(v)$; the proofs go through for $\epsilon(v) = 1/\log v$ as well. Thus the use of the χ-function is not essential for deriving Theorem 5.4.

However, a tighter analysis of the size of $seq^*(u)$ stressing $\chi(v)$ yields a better bound for the number of nondeterministic steps. In fact, we show in the next result that $O(\chi(n) \cdot \log n)$ bit guesses are sufficient. Note that $\chi(n) = o(\log n)$; thus the result is an effective improvement. Moreover, it also shows that DUAL is most likely not complete for co-β_2P.

THEOREM 5.6. *Deciding whether monotone CNFs φ and ψ are nondual is feasible in polynomial time with $O(\chi(n) \cdot \log n)$ nondeterministic steps, where $n = |\varphi| + |\psi|$.*

Proof. In the proof of Theorem 5.4, our estimates of the components of $seq^*(u)$ were rather crude. With more effort, we establish the following.

FACT C. *For any u in T, the size of $seq^*(u)$ is $O(\chi(v^*) \cdot \log(v^*))$.*

Assume node u' in T is a child of u generated via a b-rule. The j-label of the arc (u, u') serves to identify one clause of $I(u)$. Clearly, there are no more than $v(u)$ such clauses. Thus $\log v(u)$ bits suffice to represent any j-label.

Observe that if u is a node of T, then any path π from u to a node w in T contains at most $v(u)$ nodes since the volume always decreases by at least 1 in each decomposition step. Thus the number of a-labeled arcs in π is bounded by $v(u)$ and not just by v^* $(= v(r))$.

For each node u and descendant w of u in T, let

$$f(u, w) = \sum_{u' \in B(u,w)} \log v(u'),$$

where $B(u, w)$ is the set of all nodes t on the path from u to w such that the arc from t to its successor on the path is b-labeled.

By what we have observed, the total size of all encodings of j-labels in $seq^*(u)$ is at most $f(v^*, u)$, and the size of all α-blocks in $seq^*(u)$ is at most $\log(v^*)+f(v^*, u)$, where the first term takes care of the first α-block and the second of all other α-blocks. Therefore, the total size of all α-blocks and all bj-blocks in $seq^*(u)$ is $O(f(v^*, u)+\log(v^*))$.

We now show that, for each node u and descendant w of u in T, it holds that

$$f(u, w) \leq \log(v(u)) \cdot \chi(v(u)).$$

The proof is by induction on the number $|B(u, w)|$ of b-labeled arcs on the path π from u to w. If $|B(u, w)| = 0$, then obviously $f(u, w) = 0 \leq v(u)$.

Assume the claim holds for $|B(u', w)| \leq i$, and consider $|B(u, w)| = i + 1$. Let t be the first node on π contained in $B(u, w)$, and let t' be its child on π. Clearly, $f(u, w) = f(t, w)$, and thus we obtain

$$\begin{aligned}
f(u, w) &= \log(v(t)) + f(t', w) \\
&\leq \log(v(t)) + \log(v(t')) \cdot \chi(v(t')) && \text{(induction hypothesis)} \\
&\leq \log(v(t)) + (\log(v(t)) - \log(\chi(v(t)))) \cdot \chi(v(t)) \\
&&& \text{(as } v(t') \leq \tfrac{v(t)}{\chi(v(t))},\ \chi(v(t')) \leq \chi(v(t))\text{)} \\
&= \log(v(t)) \cdot \chi(v(t)) && \text{(as } \log(\chi(y)) \cdot \chi(y) = \log y \text{ for all } y\text{).}
\end{aligned}$$

Thus $f(u, w) \leq \log(v(u)) \cdot \chi(v(u))$. This concludes the induction and proves the claim.

Finally, we show that the total size of all γ blocks in $seq^*(u)$, i.e., the number of all c-labels in $seq(u)$, is bounded by $\chi(v^*) \cdot \log(v^*) < \log^2 v^*$. Indeed, assume a c-rule is applied to generate a child t' of any node t, and let $v = v(t)$, $v' = v(t')$. Since

$\epsilon_i^{\varphi} > \epsilon(v)$ and $\epsilon_i^{\psi} > \epsilon(v)$, we have $v' < (1 - \epsilon(v)) \cdot v$. Since $\chi(v^*) > \chi(v)$, we have $\epsilon(v) = 1/\chi(v) > 1/\chi(v^*)$, and thus

$$v' < \left(1 - \frac{1}{\chi(v^*)}\right) \cdot v.$$

Hence any node in T resulting after $\chi(v^*) \cdot \log(v^*)$ applications of a c-rule has volume at most

$$v^* \cdot \left(1 - \frac{1}{\chi(v^*)}\right)^{\chi(v^*)\cdot \log v^*} = v^* \cdot \left[\left(1 - \frac{1}{\chi(v^*)}\right)^{\chi(v^*)}\right]^{\log v^*} \leq v^* \cdot \left(\frac{1}{e}\right)^{\log v^*} \leq 1$$

(cf. also (9)). Consequently, along each branch in T there must be no more than $\chi(v^*) \cdot \log v^*$ applications of a c-rule. In summary, the total sizes of all α-blocks, all γ-blocks, and all encodings of j-labels in $seq^*(u)$ are all bounded by $\chi(v^*) \cdot \log v^*$. This proves Fact C.

As a consequence, nonduality of a monotone pair (φ^*, ψ^*) can be recognized in polynomial time with $O(\chi(v^*) \cdot \log v^*)$ many bit guesses. As already observed in the last lines of [17], we have $\chi(v^*) < 2\chi(n)$. Furthermore, $v^* \leq n^2$; thus $\log v^* \leq 2 \log n$. Hence nonduality of (φ^*, ψ^*) can be recognized in polynomial time with $O(\chi(n) \cdot \log(n))$ bit guesses. □

COROLLARY 5.7. *Problem* DUAL *is solvable in deterministic* $n^{O(\chi(n))}$ *time, where* $n = |\varphi| + |\psi|$.

Remark 5.1. Note that the sequence $seq(u)$ describing a path from the root of T to a "failure leaf" with label $I(u) = (\varphi', \psi')$ describes a choice of values for all variables in $V(\varphi \wedge \psi) \setminus V(\varphi' \wedge \psi')$. By completing it with values for $V(\varphi' \wedge \psi')$ that show nonduality of (φ', ψ'), which is possible in polynomial time, we obtain in polynomial time from $seq(u)$ a vector w such that $f(w) \neq g^d(w)$. It also follows from the proof of Theorem 5.6 that a witness w for $f \neq g^d$ (if one exists) can be found in polynomial time with $O(\chi(n) \cdot \log n)$ nondeterministic steps.

5.3. Application of Beigel and Fu's results. While our independently developed methods substantially differ from those in [1, 2], membership of problem DUAL in co-β_2P may also be obtained by exploiting Beigel and Fu's Theorem 8 in [1] (or, equivalently, Theorem 11 in [2]). They show how to convert certain recursive algorithms that use disjunctive self-reductions, have runtime bounded by $f(n)$, and fulfill certain additional conditions into polynomial algorithms using $\log(f(n))$ nondeterministic steps (cf. [2, Section 5]).

Let us first introduce the main relevant definitions of [1]. Let $\|y\|$ denote the size of a problem instance y.

DEFINITION 5.2 (see [1]). *A partial order* $\prec$ *(on problem instances) is polynomially well founded if there exists a polynomial bounded function* p *such that*

- $y_m \prec \cdots \prec y_1 \Rightarrow m \leq p(\|y_1\|)$ *and*
- $y_m \prec \cdots \prec y_1 \Rightarrow \|y_m\| \leq p(\|y_1\|)$.

For technical simplicity, [1] considers only languages (of problem instances) containing the empty string, Λ.

DEFINITION 5.3 (see [1]). *A disjunctive self-reduction (for short, d-self-reduction) for a language* L *is a pair* $\langle h, \prec \rangle$ *of a polynomial time computable function* $h(x) = \{x_1, \ldots, x_m\}$ *and a polynomially well-founded partial order* $\prec$ *on problem instances such that*

1. Λ *is the only minimal element under* $\prec$*;*

2. *for all* $x \neq \Lambda$, $x \in L \equiv h(x) \cap L \neq \emptyset$;
3. *for all* x, $x_i \in h(x) \Rightarrow x_i \prec x$.

DEFINITION 5.4 (see [1]). *Let* $\langle h, \prec \rangle$ *be a d-self-reduction, and let* x *be a problem instance.*

1. $T_{h,\prec}(x)$ *is the unordered rooted tree that satisfies the following rules:* (1) *the root is* x; (2) *for each* y, *the set of children of* y *is* $h(y)$.
2. $|T_{h,\prec}(x)|$ *is the number of leaves in* $T_{h,\prec}(x)$.

DEFINITION 5.5 (see [1]). *Let* T *be a polynomial time computable function. A language* L *is in* $\mathrm{REC}(T(x))$ *if there is a d-self-reduction* $\langle h, \prec \rangle$ *for* L *such that, for all* x,

1. $|T_{h,\prec}(x)| \leq T(x)$, *and*
2. $T(x) \geq \sum_{x_i \in h(x)} T(x_i)$.

Let $T(x)$-P denote the set of all (languages of) problems whose yes-instances x are recognizable in polynomial time with $T(x)$ nondeterministic bit guesses.

THEOREM 5.8 (see [1]). $\mathrm{REC}(T(x)) \subseteq \lceil \log T(x) \rceil$-P.

We now show that Theorem 5.8, together with Fredman's and Khachiyan's proof of the deterministic complexity of Algorithm B, can be used to prove that problem DUAL is in co-β_2P.

Let L denote the set of all nondual monotone pairs (φ, ψ) plus Λ. Let us identify each monotone pair (φ, ψ) which satisfies LCHECK(φ, ψ) but does not satisfy TEST(φ, ψ) with the "bottom element" Λ. Thus, if a node in the recursion tree T has a child labeled with such a pair, then the label is simply replaced by Λ.

Let us define the order $\prec$ on monotone pairs plus Λ as follows: $J \prec I$ if $I \neq J$ and either $J = \Lambda$ or J labels a node of the recursion tree generated by Algorithm B on input I. It is easy to see that both conditions of Definition 5.2 apply; therefore, $\prec$ is polynomially well founded. In fact, we may define the polynomial p by the identity function; since the sizes of the instances in the recursion tree strictly decrease on each path in T, the two conditions hold.

Define h as the function which associates with each monotone pair $I = (\varphi, \psi)$ those instances that label all children of the root by Algorithm B on input I. Clearly h satisfies all three conditions of Definition 5.3, and hence $\langle h, \prec \rangle$ is a d-self-reduction for L.

Let T be the function which to each instance I associates $v(I)^{\log v(I)}$. (Recall that $v(I)$ denotes the volume of I.) It is now sufficient to check that conditions 1 and 2 of Definition 5.5 are satisfied and to ensure that Theorem 5.8 can be applied.

That item 1 of Definition 5.5 is satisfied follows immediately from Lemma 5 in [17], which states that the maximum number of recursive calls of Algorithm B on any input I of volume v is bounded by $v^{\chi(v)}$ $(\leq v^{\log v})$. Retain, however, that the proof of this lemma is noticeably more involved than our proof of the membership of DUAL in co-β_2P.

To verify item 2 of Definition 5.5, it is sufficient to prove that for a volume $v > 4$ of any input instance to Algorithm B, it holds that

$$v^{\log v} \geq (v-1)^{\log(v-1)} + \frac{v}{3} \cdot \left(\frac{v}{2}\right)^{\log \frac{v}{2}}, \quad \text{and} \tag{10}$$

$$v^{\log v} \geq 2(\alpha \cdot v)^{\log(\alpha \cdot v)}, \quad \text{where } \alpha = 1 - 1/\log v; \tag{11}$$

here, (10) arises from the rules (i), (ii), and (11) from rule (iii). As for (10), the child of u from (i.a), respectively, (ii.a), has volume at most $v - 1$, and there are at most $v/3$ many children from (i.b), respectively, (ii.b), since $\min(|\varphi|, |\psi|) > 2$ (recall that

$v = |\varphi| \cdot |\psi|$); furthermore, each such child has volume $\leq \epsilon(v) \cdot v \leq \frac{1}{2}v$. In the case of (11), the volume of each child of u is bounded by $(1 - \epsilon(v)) \cdot v \leq (1 - 1/\log v) \cdot v$; note also that $v^{\log v}$ monotonically increases for $v > 4$. To see (10), we have

$$\begin{aligned}(v-1)^{\log(v-1)} + \tfrac{v}{3} \cdot (\tfrac{v}{2})^{\log \frac{v}{2}} &\leq (v-1)^{\log v} + \tfrac{v}{3} \cdot \tfrac{v^{\log v - 1}}{2^{\log v - 1}} \\ &= v^{\log v} \cdot (1 - \tfrac{1}{v})^{\log v} + \tfrac{2 \cdot v^{\log v}}{3 \cdot v} \\ &\leq v^{\log v} \cdot (1 - \tfrac{1}{v} + \tfrac{2}{3 \cdot v}) \\ &= v^{\log v} \cdot (1 - \tfrac{1}{3 \cdot v}) \\ &< v^{\log v};\end{aligned}$$

to show (11), note that

$$\begin{aligned}2(\alpha \cdot v)^{\log(\alpha \cdot v)} &= 2\alpha^{\log v + \log \alpha} \cdot v^{\log v + \log \alpha} \\ &\leq 2(\tfrac{1}{e} \cdot \alpha^{\log \alpha}) \cdot v^{\log v + \log \alpha} \qquad (\alpha^{\log v} \leq 1/e \text{ by (9)}) \\ &= \tfrac{2}{e} \cdot (\alpha \cdot v)^{\log \alpha} \cdot v^{\log v} \\ &\leq \tfrac{2}{e} \cdot v^{\log v} \qquad ((\alpha \cdot v)^{\log \alpha} \leq 1, \text{ i.e., } \log \alpha \cdot (\log \alpha + \log v) \leq 0, \\ &\qquad\qquad \text{since } -1 < \log \alpha \leq 0 \text{ and } \log v > 2) \\ &< v^{\log v}.\end{aligned}$$

We can thus apply Theorem 5.8 and conclude that the complement of DUAL is in $\lceil \log T(x) \rceil$-P and thus also in β_2P.

The advantage of Beigel and Fu's method is its very abstract formulation. The method has two disadvantages, however, that are related to the two items of Definition 5.5.

The first item requires that $T(x)$ is at least the number of leaves in the tree for x. In order to show this, one must basically prove a deterministic time bound for the considered algorithm (or at least a bound of the number of recursive calls for each instance, which is often tantamount to a time bound). The method does not suggest how to do this but presupposes that such a bound exists. (In the present case, this was done by Fredman and Khachiyan in a nontrivial proof.) The second item requires proving that the T-value of any node x in the recursion tree is at least the sum of the T-values of its children. This may be hard to show in many cases and does not necessarily hold for every upper bound T.

Our method instead does not require an a priori time bound but directly constructs a nondeterministic algorithm from the original deterministic algorithm, which lends itself to a simple analysis that directly leads to the desired nondeterministic time bound. The deterministic time bound follows as an immediate corollary. It turns out (as exemplified by the very simple proof of Theorem 5.4) that the analysis involved in our method can be simpler than an analysis according to previous techniques.

6. Conclusion. We have presented several new cases of the monotone dualization problem which are solvable in output polynomial time. These cases generalize some previously known output polynomial cases. Furthermore, we have shown by rather simple means that nondual monotone pairs (φ, ψ) can be recognized, using a nondeterministic variant of Fredman and Khachiyan's Algorithm B [17] in polynomial time with $O(\log^2 n)$ many bit guesses, which places problem DUAL in the class co-β_2P. In fact, a refined analysis revealed that this is feasible in polynomial time with $O(\chi(n) \cdot \log n) = O(\log^2 n / \log \log n)$ many bit guesses.

While our results document progress on DUAL and DUALIZATION and reveal novel properties of these problems, the question of whether dualization of monotone pairs (φ, ψ) is feasible in polynomial time remains open. It would be interesting to see whether the amount of guessed bits can be further significantly decreased, e.g., to $O(\log \log n \cdot \log n)$ many bits.

Note added in proof. Independently, Dimitris J. Kavvadias and Elias C. Stavropoulos have shown that DUALIZATION can be solved with limited nondeterminism using $O(\log^2 n)$ deterministic steps in polynomial time (see [31]).

Acknowledgments. Georg Gottlob thanks José Balcázar for inviting him to an interesting workshop in Bellaterra, Barcelona, Spain and thanks Leonard Pitt for giving an elucidating lecture on the dualization problem there. The authors appreciate the review comments which helped to improve the paper.

REFERENCES

[1] R. BEIGEL AND B. FU, *Molecular computing, bounded nondeterminism, and efficient recursion*, in Proceedings of the 24th International Colloquium on Automata, Languages, and Programming (ICALP '97), Lecture Notes in Comput. Sci. 1256, Springer-Verlag, New York, 1997, pp. 816–826.

[2] R. BEIGEL AND B. FU, *Molecular computing, bounded nondeterminism, and efficient recursion*, Algorithmica, 25 (1999), pp. 222–238.

[3] C. BIOCH AND T. IBARAKI, *Complexity of identification and dualization of positive Boolean functions*, Inform. and Comput., 123 (1995), pp. 50–63.

[4] E. BOROS, K. ELBASSIONI, V. GURVICH, AND L. KHACHIYAN, *An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension*, Parallel Process. Lett., 10 (2000), pp. 253–266.

[5] E. BOROS, K. ELBASSIONI, V. GURVICH, L. KHACHIYAN, AND K. MAKINO, *Dual-bounded generating problems: All minimal integer solutions for a monotone system of linear inequalities*, SIAM J. Comput., 31 (2002), pp. 1624–1643.

[6] E. BOROS, V. GURVICH, AND P. L. HAMMER, *Dual subimplicants of positive Boolean functions*, Optim. Methods Softw., 10 (1998), pp. 147–156.

[7] E. BOROS, V. GURVICH, L. KHACHIYAN, AND K. MAKINO, *Dual-bounded generating problems: Partial and multiple transversals of a hypergraph*, SIAM J. Comput., 30 (2001), pp. 2036–2050.

[8] E. BOROS, V. GURVICH, L. KHACHIYAN, AND K. MAKINO, *On the complexity of generating maximal frequent and minimal infrequent sets*, in Proceedings of the 19th Symposium on Theoretical Aspects of Computing (STACS '99), Lecture Notes in Comput. Sci. 2285, Springer-Verlag, New York, 2002, pp. 133–141.

[9] E. BOROS, P. L. HAMMER, T. IBARAKI, AND K. KAWAKAMI, *Polynomial-time recognition of 2-monotonic positive Boolean functions given by an oracle*, SIAM J. Comput., 26 (1997), pp. 93–109.

[10] N. M. C. DOMINGO AND L. PITT, *Efficient read-restricted monotone CNF/DNF dualization by learning with membership queries*, Machine Learning, 37 (1999), pp. 89–110.

[11] C. CHEKURI AND A. RAJARAMAN, *Conjunctive query containment revisited*, in Proceedings of the 6th International Conference on Database Theory (ICDT-97), Lecture Notes in Comput. Sci. 1186, F. Afrati and P. Kolaitis, eds., Springer-Verlag, New York, 1997, pp. 56–70.

[12] Y. CRAMA, *Dualization of regular Boolean functions*, Discrete Appl. Math., 16 (1987), pp. 79–85.

[13] C. DOMINGO, *personal communication*, Department of Computer Science, Tokyo Institute of Technology, Meguro-ku Ookayama, Tokyo, Japan, 1997.

[14] T. EITER, *Exact transversal hypergraphs and application to Boolean μ-functions*, J. Symbolic Comput., 17 (1994), pp. 215–225.

[15] T. EITER AND G. GOTTLOB, *Identifying the minimal transversals of a hypergraph and related problems*, SIAM J. Comput., 24 (1995), pp. 1278–1304. Extended paper: Technical report CD-TR 91/16, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1991.

[16] R. Fagin, *Degrees of acyclicity for hypergraphs and relational database schemes*, J. ACM, 30 (1983), pp. 514–550.
[17] M. Fredman and L. Khachiyan, *On the complexity of dualization of monotone disjunctive normal forms*, J. Algorithms, 21 (1996), pp. 618–628.
[18] H. Garcia-Molina and D. Barbara, *How to assign votes in a distributed system*, J. ACM, 32 (1985), pp. 841–860.
[19] D. Gaur, *Satisfiability and Self-Duality of Monotone Boolean Functions*, Dissertation, School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada, 1999.
[20] D. Gaur and R. Krishnamurti, *Self-duality of bounded monotone Boolean functions and related problems*, in Proceedings of the 11th International Conference on Algorithmic Learning Theory (ALT), Lecture Notes in Comput. Sci. 1968, Springer-Verlag, New York, 2000, pp. 209–223.
[21] G. Gogic, C. Papadimitriou, and M. Sideri, *Incremental recompilation of knowledge*, J. Artificial Intelligence Res., 8 (1998), pp. 23–37.
[22] J. Goldsmith, M. Levy, and M. Mundhenk, *Limited nondeterminism*, SIGACT News, 27 (1996), pp. 20–29.
[23] G. Gottlob, N. Leone, and F. Scarcello, *Hypertree decompositions and tractable queries*, J. Comput. System Sci., 64 (2002), pp. 579–627.
[24] M. Graham, *On the Universal Relation*, Technical report, University of Toronto, Toronto, Ontario, Canada, 1979.
[25] D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen, *Data mining, hypergraph transversals, and machine learning*, in Proceedings of the 16th ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-96), ACM, New York, 1993, pp. 209–216.
[26] V. A. Gurvich, *Nash-solvability of games in pure strategies*, USSR Comput. Math. and Math. Phys., 15 (1975), pp. 357–371 (in Russian).
[27] T. Ibaraki and T. Kameda, *A theory of coteries: Mutual exclusion in distributed systems*, IEEE Trans. Parallel Distrib. Systems, 4 (1993), pp. 779–794.
[28] D. S. Johnson, *Open and Closed Problems in NP-Completeness*, Lecture given at the International School of Mathematics G. Stampacchia Summer School, NP-Completeness: The First 20 Years, Erice, Sicily, Italy, 1991.
[29] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou, *On generating all maximal independent sets*, Inform. Process. Lett., 27 (1988), pp. 119–123.
[30] D. J. Kavvadias, C. Papadimitriou, and M. Sideri, *On Horn envelopes and hypergraph transversals*, in Proceedings of the 4th International Symposium on Algorithms and Computation (ISAAC-93), Lecture Notes in Comput. Sci. 762, W. Ng, ed., Springer-Verlag, New York, 1993, pp. 399–405.
[31] D. J. Kavvadias and E. C. Stavropoulos, *Monotone Boolean dualization is in co-NP*$[\log^2 n]$, Inform. Process. Lett., 85 (2003), pp. 1–6.
[32] R. Khardon, *Translating between Horn representations and their characteristic models*, J. Artificial Intelligence Res., 3 (1995), pp. 349–372.
[33] C. M. R. Kintala and P. C. Fischer, *Refining nondeterminism in relativized polynomial-time bounded computations*, SIAM J. Comput., 9 (1980), pp. 46–53.
[34] E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, *Generating all maximal independent sets: NP-hardness and polynomial-time algorithms*, SIAM J. Comput., 9 (1980), pp. 558–565.
[35] L. Lovász, *Combinatorial Optimization: Some Problems and Trends*, Technical report, DIMACS 92-53, RUTCOR, Rutgers University, New Brunswick, NJ, 1990.
[36] K. Makino, *Efficient dualization of* $O(\log n)$*-term monotone disjunctive normal forms*, Discrete Appl. Math., 126 (2003), pp. 305–312.
[37] K. Makino and T. Ibaraki, *The maximum latency and identification of positive Boolean functions*, SIAM J. Comput., 26 (1997), pp. 1363–1383.
[38] K. Makino and T. Ibaraki, *A fast and simple algorithm for identifying* 2*-monotonic positive Boolean functions*, J. Algorithms, 26 (1998), pp. 291–305.
[39] H. Mannila and K.-J. Räihä, *Design by example: An application of Armstrong relations*, J. Comput. System Sci., 22 (1986), pp. 126–141.
[40] C. Papadimitriou, *NP-completeness: A retrospective*, in Proceedings of the 24th International Colloquium on Automata, Languages, and Programming (ICALP '97), Lecture Notes in Comput. Sci. 1256, Springer-Verlag, New York, 1997, pp. 2–6.
[41] U. Peled and B. Simeone, *An* $O(nm)$*-time algorithm for computing the dual of a regular Boolean function*, Discrete Appl. Math., 49 (1994), pp. 309–323.

[42] N. M. L. Pitt, *Generating all maximal independent sets of bounded-degree hypergraphs*, in Proceedings of the 10th Annual Conference on Computational Learning Theory (COLT), Nashville, TN, 1997, pp. 211–217.
[43] K. Ramamurthy, *Coherent Structures and Simple Games*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990.
[44] R. Read, *Every one a winner, or how to avoid isomorphism when cataloging combinatorial configurations*, Discrete Appl. Math., 2 (1978), pp. 107–120.
[45] R. Reiter, *A theory of diagnosis from first principles*, Artificial Intelligence, 32 (1987), pp. 57–95.
[46] N. Robertson and P. Seymour, *Graph minors* II: *Algorithmic aspects of tree-width*, J. Algorithms, 7 (1986), pp. 309–322.
[47] K. Takata, *On the sequential method for listing minimal hitting sets*, in Proceedings of the Workshop on Discrete Mathematics and Data Mining, 2nd SIAM International Conference on Data Mining, Arlington, VA, 2002.
[48] H. Tamaki, *Space-efficient enumeration of minimal transversals of a hypergraph*, IPSJ-AL, 75 (2000), pp. 29–36.
[49] R. E. Tarjan and M. Yannakakis, *Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, SIAM J. Comput., 13 (1984), pp. 566–579.
[50] V. D. Thi, *Minimal keys and antikeys*, Acta Cybernet., 7 (1986), pp. 361–371.
[51] B. Toft, *Colouring, stable sets and perfect graphs*, in Handbook of Combinatorics, Vol. 1, R. Graham, M. Grötschel, and L. Lovász, eds., Elsevier, New York, 1996, ch. 4.
[52] C. T. Yu and M. Ozsoyoglu, *An algorithm for tree-query membership of a distributed query*, in Proceedings of the IEEE Conference on Computer and Software Applications, IEEE Computer Society, Los Alamitos, CA, 1979, pp. 306–312.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 538–547

COMPUTING THE MEDIAN WITH UNCERTAINTY*

TOMÁS FEDER†, RAJEEV MOTWANI‡, RINA PANIGRAHY§, CHRIS OLSTON¶, AND JENNIFER WIDOM¶

Abstract. We consider a new model for computing with uncertainty. It is desired to compute a function $f(X_1, \ldots, X_n)$, where $X_1, \ldots, X_n$ are unknown but guaranteed to lie in specified intervals $I_1, \ldots, I_n$. It is possible to query the precise value of any X_j at a cost c_j. The goal is to pin down the value of f to within a precision δ at a minimum possible cost. We focus on the selection function f which returns the value of the kth smallest argument. We present optimal offline and online algorithms for this problem.

Key words. median, selection, uncertainty, query processing, online algorithms

AMS subject classifications. 68W25, 68P99

PII. S0097539701395668

1. Introduction.

Consider the following model for computing with uncertainty. We wish to compute a function $f(X_1, \ldots, X_n)$ over n real-valued arguments. The values of the variables $X_1, \ldots, X_n$ are not known in advance; however, we are provided with real intervals $I_1, \ldots, I_n$ along with a guarantee that, for each j, $X_j \in I_j$. Furthermore, it is possible to query the true value x_j of each X_j at a cost c_j. The goal is to pin down the value of f into an interval of size $\delta \geq 0$. Thus we are faced with the following optimization problem: *Given a function f, precision parameter δ, real intervals $I_1, \ldots, I_n$, and query costs $c_1, \ldots, c_n$, pin down the value of f to an interval of size δ using a set of queries of minimum total cost.* Note that there are two natural versions of this problem: in the *online* version, the sequence of queries is chosen *adaptively* in that each successive query is answered before the next one is chosen; and, in the *offline* version, where the entire set of queries must be specified completely before the answers are provided, it must be guaranteed that f can be pinned down as desired regardless of the results of the queries.

This model is motivated by the work of Olston and Widom [4] on query processing over replicated databases, where local cached copies of databases are used to support quick processing of queries at client sites. Each data value cached at a client has a corresponding master copy maintained by a remote database where all the updates take place. The frequency of updates makes it infeasible to maintain consistency between the cached copies and master copies, and the data values in the cache are likely to become stale and drift from the master values. However, the clients store for

*Received by the editors September 27, 2001; accepted for publication (in revised form) October 24, 2002; published electronically February 20, 2003. A preliminary version of this paper has appeared as an extended abstract in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing,* 2000.

http://www.siam.org/journals/sicomp/32-2/39566.html

†Department of Computer Science, Stanford University, Stanford, CA 94305 (tomas@theory.stanford.edu).

‡Department of Computer Science, Stanford University, Gates Building 4B, Stanford, CA 94305 (rajeev@ cs.stanford.edu). This author was supported by ARO MURI grant DAAH04-96-1-0007 and NSF grants IIS-9811904 and IIS-0118173.

§Cisco Systems, 170 West Tasman Drive, San Jose, CA 95134 (rinap@cisco.com).

¶Department of Computer Science, Stanford University, Gates Building 4A, Stanford, CA 94305 (olston@cs.stanford.edu, widom@cs.stanford.edu). C. Olston was supported by NSF grant IIS-9811947 and an NSF graduate research fellowship. J. Widom was supported by NSF grants IIS–9811947 and IIS–0118173.

each cached data value an interval that is guaranteed to contain the master value. In processing an aggregation query at a client cache, it is desired to compute a function f defined over the data values to within a specified precision δ. For each data value X_j, it is possible to query the master copy for the exact value, incurring a cost, rather than use the interval I_j at the cached copy. In some cases, the cost to query the master copy of any data value is the same, but in other cases, the master copies are spread across multiple remote databases, each with a different access cost that is typically predictable and static. In general, querying the master copy of each data value X_j incurs a potentially different static cost c_j. The goal then is to compute the result of an aggregation query to within the desired precision at the minimum possible total cost. Systems considerations sometimes make it desirable to perform all queries to the master copy en masse, motivating the offline version of the problem.

Olston and Widom [4] considered functions f which are simple aggregation functions, including

$$\begin{aligned} \mathrm{SUM}\left(f(X_1,\ldots,X_n) = \sum_{j=1,n} X_j\right),\\ \mathrm{MIN}(f(X_1,\ldots,X_n) = \min_{j=1,n} X_j), \text{ and}\\ \mathrm{MAX}(f(X_1,\ldots,X_n) = \max_{j=1,n} X_j). \end{aligned}$$

It was observed that the SUM problem is isomorphic to the knapsack problem [2]: Consider the set of X_j's that are *not* queried; clearly, the sum of the corresponding interval sizes must be at most δ, and the objective function is equivalent to maximizing the corresponding costs. The case of the selection function, i.e., where the function f returns the value of its kth smallest argument, and particularly the median, was left open.

In this paper, we resolve the complexity of the problem of computing the median (and, in general, the kth smallest element) under the model of computing with uncertainty, in both the offline and the online settings. We begin by expressing the offline selection problem in terms of an integer linear program (section 3). Not only is this integer program's structure critical to the development of our offline algorithms, but it also helps provide a useful lower bound for the online selection problem. Based on these insights, we provide a polynomial-time algorithm for the offline case with unit costs and a general offline algorithm with running time exponential in k (section 4). Then we apply this tool to the development of an optimal online algorithm and provide a tight relationship between the performance of the optimal online and offline algorithms (section 5). We extend our results to obtaining a polynomial-time algorithm for the general offline case based on the use of linear programming (section 6). We also present a simple approximation algorithm that does not rely on linear programming. Finally, we define a class of problems called *interval problems* and show that this is equivalent to the offline median problem and includes weighted bipartite matching as a special case (section 7). It also turns out that our offline selection problem can be expressed as a min-cost network flow problem (section 7). Finally, we demonstrate that a mild generalization of our selection problem is NP-hard.

2. Preliminaries. An instance of the *selection problem* consists of n intervals $I_1, I_2, \ldots, I_n$, an associated real cost $c_j \geq 0$ for each interval $I_j = [l(I_j), r(I_j)]$, an integer $1 \leq k \leq \lceil \frac{n}{2} \rceil$, and a value $\delta \geq 0$. Each interval I_j has an unknown point p_j. The aim is to estimate the value of the kth smallest p_j with precision δ. An algorithm

can query an interval I_j, paying cost c_j, and obtain the point p_j. The interval I_j is then replaced with the interval $I'_j = [p_j, p_j]$. At any point in time, for the current intervals I_j, the kth smallest point can be pinned down into an interval $[l, r]$.

LEMMA 2.1. *Given the intervals I_j, the kth smallest point must lie in the interval $[l, r]$, where l is the kth smallest $l(I_j)$, and r is the $(n-k+1)$th largest $r(I_j)$.*

Note that this is the smallest interval to which the kth smallest point can be pinned, in the absence of any other information.

When the algorithm terminates, we must have $r - l \leq \delta$. We seek an algorithm that minimizes the worst-case total cost to achieve this bound. In the unit-cost case, all $c_j = 1$, and the aim is to minimize the number of intervals queried. Two special cases of interest seek the smallest element, with $k = 1$, or the median of the elements, with n odd and $k = \lceil \frac{n}{2} \rceil$. The special case in which $k = 1$ has been addressed in [4] and turns out to have linear complexity. There are two variants of the problem we are interested in. In the online version of the problem that was stated above, we can use the points returned by previous queries to decide which interval to query next. We are also interested in the offline problem, where the algorithm must decide which intervals to query at the same time and guarantee that, regardless of the points obtained in these queries, the estimate $[l, r]$ will have $r - l \leq \delta$.

3. An integer programming formulation. We shall express the offline problem as an integer linear program. The structure of this integer program is critical to the development of our algorithms; also, we obtain from the constraints of this integer program a lower bound on the worst-case online cost. The integer program has a variable $x_j \in \{0, 1\}$ that expresses whether the interval I_j is queried. The aim is then to minimize $\sum c_j x_j$. To describe the constraints, we introduce some terminology. Consider an interval $O = [l(O), r(O)]$ of size $|O| = r(O) - l(O) > \delta$. Let a be the number of I_j with $l(I_j) \leq l(O)$, and let b be the number of I_j with $r(I_j) \geq r(O)$. If $a \geq k$ and $b \geq n - k + 1$, then we say that O is an *obstruction*. We shall show that, for all obstructions O, the offline algorithm must satisfy $\sum x_j \geq a + b - n$, where the sum is over the x_j such that I_j contains O. That is, at least $a + b - n$ intervals containing O must be queried in order to guarantee the bound δ on the final estimate.

To obtain a finite number of constraints, we introduce some additional terminology. A *proper obstruction* is an obstruction O such that, for some input intervals $I_j, I_{j'}$, we have $l(O) = l(I_j)$ and $r(O) = r(I_{j'})$. A *minimal proper obstruction* is a proper obstruction that does not contain any other proper obstructions. The integer program is then to minimize $\sum c_j x_j$, with $x_j \in \{0, 1\}$, subject to the constraint that for each minimal proper obstruction O_i, $\sum x_j \geq a_i + b_i - n$, where the sum is over the x_j such that I_j contains O_i. Notice that there are at most k minimal proper obstructions, since $r(O_i)$ must be one of the k smallest $r(I_j)$.

It will sometimes be convenient, for an obstruction O, to write $a + b - n = d - e$, where d is the number of I_j containing O, and e is the number of I_j inside O, that is, with $l(I_j) > l(O)$ and $r(I_j) < r(O)$. To see why this equality holds, write it as $n = a + b - d + e$; that is, the n intervals can be counted as a intervals with $l(I_j) \leq l(O)$ and b intervals with $r(I_j) \geq r(O)$, subtracting the d intervals that satisfy both conditions and adding the e intervals that satisfy neither condition.

For the online problem, we let V be the maximum over all minimal proper obstructions O_i of the minimum of $\sum c_j x_j$ with $\sum x_j = a_i + b_i - n$, where the sums are over the x_j with I_j containing O_i. That is, for each minimal proper obstruction O_i, we determine the sum of the $(a_i + b_i - n)$ smallest costs of intervals containing O_i and find the worst such O_i.

PROPOSITION 3.1. *The integer program solves the offline selection problem. The quantity V is a lower bound on the maximum total cost for the online selection problem.*

Proof. Say that a choice of queried intervals I_j *clears* an obstruction O if, regardless of the points p_j returned for the queried intervals, the interval O will no longer be an obstruction after the queried I_j are replaced by $I'_j = [p_j, p_j]$. We show that O is cleared if and only if $\sum x_j \geq a + b - n$, where the sum is over the intervals I_j containing O.

If $\sum x_j \leq a + b - n - 1 = (a - k) + (b - (n - k + 1))$, then for each I_j queried not containing O, we can return $p_j = l(I_j)$ if $r(I_j) < r(O)$ and return $p_j = r(I_j)$ otherwise, with $l(I_j) > l(O)$. Clearly, this choice of p_j does not decrease the values a, b for O. For the I_j queried containing O, we can return $p_j = l(I_j)$ for at most $b - (n - k + 1)$ of them and return $p_j = r(I_j)$ for at most $a - k$ of them. Then O will still be an obstruction, since there will be still at least k intervals with $l(I'_j) \leq l(O)$, and at least $n - k + 1$ intervals with $r(I'_j) \geq r(O)$. That is, the obstruction is not cleared.

For the converse, suppose $\sum x_j \geq a + b - n = (a - k) + (b - (n - k + 1)) + 1$. Then in each queried interval I_j containing O, either a decreases by 1 if $p_j > l(O)$ or b decreases by 1 if $p_j < r(O)$. Thus, in the end, we will have either $a' < k$ or $b' < n - k + 1$, so O will no longer be an obstruction. That is, the obstruction is cleared. This completes the proof of the equivalence.

A choice of intervals to be queried solves the offline problem if and only if it clears all obstructions. We show that it is sufficient to consider minimal proper obstructions. For an obstruction O, the smallest proper obstruction O' containing it has $a' = a$, $b' = b$, and the intervals I_j containing O are the same as those containing O'. Therefore, the linear constraints for O and O' are the same, and it is sufficient to consider proper obstructions. If an obstruction is cleared, then every obstruction containing it is also cleared, so it is sufficient to consider minimal proper obstructions. This completes the proof of the characterization of the offline problem as an integer program.

We prove the lower bound V for the online problem. Let O be an obstruction giving the value V. In the proof that if $\sum x_j \leq a + b - n - 1$, then the obstruction is not cleared, we could have chosen the points returned for the queries one by one, as the queried intervals are chosen, that is, in an online fashion. Therefore, we must have $\sum x_j \geq a + b - n$, paying for the $a + b - n$ intervals of least cost containing O. ∎

4. Offline problem with unit costs. We now show how to solve the offline problem efficiently, in the unit-cost case, using the integer program described above. List the minimal proper obstructions in order, letting $O_1 < O_2$ if $l(O_1) < l(O_2)$ and $r(O_1) < r(O_2)$. For the leftmost minimal proper obstruction O_1, since all the intervals I_j containing it have the same cost, we can greedily select I_j with $r(I_j)$ as large as possible, so as to cover as many O_j as possible with it, until we have chosen $a_1 + b_1 - n$ intervals containing O_1. We then move on to O_2, with O_1 already covered, and choose additional intervals to cover O_2 as needed, again with their right endpoint as far to the right as possible, until we have chosen at least $a_2 + b_2 - n$ intervals to cover O_2. We proceed in this fashion, satisfying the constraints on minimal proper obstructions in order, from left to right.

THEOREM 4.1. *In the unit-cost case, the integer program that characterizes the offline selection problem can be solved in polynomial time by a greedy algorithm.*

This approach does not seem to work in the general offline case with arbitrary

costs, since longer intervals might have a larger cost, so that it is not clearly an advantage to choose them. However, we can obtain an exponential-time algorithm as follows. Let r be the $(n-k+1)$th largest $r(I_j)$. At most $k-1$ intervals I_j have $r(I_j) < r$. Call these the special intervals, and choose which ones will be queried in all possible ways, that is, 2^{k-1} ways. All obstructions have $r(O) \leq r$, so we can again examine the minimal proper obstructions in order from left to right, covering them with intervals I_j of least possible cost which are not special, using as many intervals as needed. The right endpoint of intervals that are not special does not matter, since it is at least r.

This gives an algorithm with a time bound of $2^k poly(n)$. A tighter time bound follows from observing that there are at most k minimal proper obstructions O_i, and by writing $a_i + b_i - n = d_i - e_i$, we observe that the number e_i of intervals inside O_i is at most $k-1$, so at most $k-1$ intervals containing O_i will not be queried. We can then just identify the at most $k-1$ intervals of largest cost that are not special and have O_i as the first minimal proper obstruction they cover. That is, after $poly(n)$ preprocessing time, each of the 2^{k-1} cases involves at most k obstructions and k^2 identified intervals, so it uses $poly(k)$ time.

THEOREM 4.2. *With arbitrary costs, the integer program that characterizes the offline selection problem can be solved in time* $poly(n) + 2^k poly(k)$.

A different approach will later enable us to obtain a polynomial-time algorithm for the general offline case.

5. The online problem. We now turn to the online problem and consider an algorithm for the general case with arbitrary costs. The greedy algorithm is as follows. Determine the interval $[l, r]$, where l is the kth smallest $l(I_j)$ and r is the $(n-k+1)$th largest $r(I_j)$. If $r - l \leq \delta$, then we are done. Otherwise, $[l, r]$ is an obstruction O. Query the I_j containing O of least cost. Once the point p_j is obtained, replace I_j by $I'_j = [p_j, p_j]$, and go back to the beginning of the algorithm.

Consider an execution of the greedy algorithm. Let O_1, O_2,..., O_s be the sequence of obstructions obtained, with O_t containing O_{t+1}. For the last obstruction O_s, let H be the set of the $a_s + b_s - n$ intervals of least cost containing O_s. Clearly, the total cost of H is at most V, the lower bound established in section 6. We show that the interval queried at stage t with $1 \leq t \leq s$ is in H, completing the proof.

The proof is by induction on t. Since the intervals in H clear the obstruction O_s, they also clear the obstruction O_t which contains O_s. In fact, the intervals in H containing O_t clear the obstruction O_t because only containing intervals matter in clearing an obstruction. By inductive hypothesis, the $t-1$ intervals previously queried are in H. However, these $t-1$ intervals have not cleared O_t. Therefore, H must have some interval containing O_t other than the $t-1$ intervals previously queried, and such an interval in H of least cost c will be queried. In the case in which there are also intervals of cost c not in H (that is, c is the largest cost in H), then an interval of cost c not in H may be queried; however, in that case, we can change H by switching this interval for the interval of cost c containing O_t in H without increasing the total cost of H.

THEOREM 5.1. *With arbitrary costs, the greedy polynomial online selection algorithm achieves the cost* V *of Proposition* 3.1, *and is therefore optimal.*

We will now compare the performance of the offline algorithm with the worst-case performance of the online algorithm, both in the unit-cost case and in the general case with arbitrary costs. For the unit-cost case, we will transform the problem as follows.

Let P be an instance of the problem. Suppose P contains two intervals I_1 and I_2

with I_2 inside I_1, that is, $l(I_1) < l(I_2)$ and $r(I_2) < r(I_1)$. We construct a new instance P' by replacing these two intervals by $I_1' = [l(I_1), r(I_2)]$ and $I_2' = [l(I_2), r(I_1)]$. We repeatedly perform this transformation until we obtain an instance $\widehat{P}$ with no interval inside another interval.

We first look at the offline problem. Consider the optimal solution for P'. If both I_1' and I_2' are queried in this solution, obtain a candidate solution for P by querying both I_1 and I_2. If only one of I_1' or I_2' is queried, then query only I_1. If neither I_1' nor I_2' is queried, then query neither I_1 nor I_2.

Since P and P' have the same left and right endpoints of intervals, the minimal proper obstructions O are the same for P and P' and have the same value of $a+b-n$. Suppose O is such an obstruction. If both I_1' and I_2' contain O, then both I_1 and I_2 contain O; and if only one of I_1' and I_2' contain O, then I_1 contains O. Therefore, the candidate solution for P is indeed a solution with the same cost (number of queried intervals) as the solution for P'.

Consider now the online problem. Here, the worst-case performance is given by the minimal proper obstruction O with the largest value $a + b - n$. This value, as we said before, is the same for P and P'.

PROPOSITION 5.2. *Consider the unit-cost case. For the offline selection problem, the performance on $\widehat{P}$ is at least as bad as the performance on P. For the online selection problem, the worst-case performance is the same for $\widehat{P}$ as for P.*

We can now compare the performance of the optimal offline algorithm with the worst-case performance of the optimal online algorithm.

THEOREM 5.3. *The worst-case performance ratio between offline and online selection algorithms is $\frac{2k-1}{k} < 2$ in the unit-cost case. Therefore, both algorithms have the same performance for the smallest element problem, while the ratio for the median problem is $\frac{2n}{n+1}$. In the case of arbitrary costs, the worst-case ratio equals k.*

Proof. We begin with the unit-cost case. By the preceding proposition, it is sufficient to consider an instance $\widehat{P}$ with no interval inside another interval. The intervals can then be ordered from left to right, breaking ties between identical intervals arbitrarily, and letting otherwise $I_1 < I_2$ if $l(I_1) \le l(I_2)$ and $r(I_1) < r(I_2)$ or if $l(I_1) < l(I_2)$ and $r(I_1) \le r(I_2)$.

Let $I_1, I_2, \ldots, I_n$ be the intervals in this order. Consider the interval I_k. Clearly I_k is also the largest obstruction, and so all minimal proper obstructions are contained in it. We can assume $|I_k| > \delta$; otherwise, no queries are needed. Since no interval is inside an obstruction, all intervals containing an obstruction will be queried. Let I_{k-s} be the first I_j with $j \le k$ such that $r(I_j) - l(I_k) > \delta$. Similarly, let I_{k+t} be the last I_j with $j \ge k$ such that $r(I_k) - l(I_j) > \delta$. Clearly $0 \le s \le k-1$ and $0 \le t \le n-k$.

The intervals preceding I_{k-s} do not contain any obstruction and will therefore not be queried by either algorithm. Similarly, the intervals following I_{k+t} will not be queried by either algorithm. The intervals from I_{k-s} to I_{k+t} contain at least one obstruction and will therefore be queried by the offline algorithm. Therefore, the offline algorithm makes $s + t + 1$ queries.

For the online algorithm, it is sufficient to consider the minimal proper obstruction with the largest number of intervals containing it. The first minimal proper obstruction is contained at least in the intervals from I_{k-s} to I_k. The last minimal proper obstruction is contained at least in the intervals from I_k to I_{k+t}. Thus the online algorithm makes at least $\max(s,t)+1$ queries in the worst case, and this quantity is the worst case when the I_j with $j < k$ do not intersect the I_j with $j > k$.

The ratio $\frac{s+t+1}{\max(s,t)+1}$ with $0 \le s \le k-1$ and $0 \le t \le n-k$ is maximized at

$s = t = k - 1$, and it then equals $\frac{2k-1}{k}$.

Consider next the case of arbitrary costs. Since the performance of the online algorithm is given by the worst constraint for a single minimal proper obstruction and there are at most k such obstructions, the ratio is at most k. An example that achieves k has $n = 2k - 1$ and consists of k disjoint intervals I_j of unit-cost plus $k - 1$ intervals I'_j of zero cost, with the I'_j containing all the I_j. The offline algorithm will have to query all the intervals, incurring cost k with the intervals I_j. The online algorithm queries the I'_j first and, depending on the resulting answers, determines which single I_j to query, with total cost 1. □

6. Offline problem with arbitrary costs. The earlier algorithm for the offline selection problem with arbitrary costs has complexity exponential in k. We now provide a polynomial-time algorithm for this problem, but this algorithm is noncombinatorial and relies on linear programming.

The polytope of the offline selection problem is defined by replacing in the integer program the conditions $x_j \in \{0, 1\}$ with linear constraints $0 \leq x_j \leq 1$, with the remaining linear constraints being the same.

THEOREM 6.1. *The vertices of the polytope of the offline selection problem are all integer vertices (i.e., 0–1 vertices). Therefore, with arbitrary costs, the problem can be solved in polynomial time by linear programming.*

Proof. Consider a vertex x of the polytope. If some x_j is either 0 or 1 for x, then use this value in the constraints where x_j occurs. We are thus left with some h variables with $0 < x_j < 1$. Since x is a vertex of the polytope, there must be some h constraints satisfied with equality that define x uniquely. The corresponding h-by-h square matrix M is a 0–1 matrix M, with the property that, for every column of M, the value 1 occurs in consecutive rows, since a variable x_j occurs in consecutive linear constraints. We show that the determinant of such a matrix M is either 0, 1, or -1. Therefore, the solution must be an integer; that is, all x_j for the vertex x are either 0 or 1, completing the proof of optimality.

Assume M has a nonzero determinant. Then some column must have a 1 in the first row. Consider the column with a 1 in the first row that has the least number r of 1's; say it is the first column. Then the 1's in the first column occur in rows $1, 2, \ldots, r$. After subtracting the first row from rows $2, \ldots, r$, we can remove the first row and the first column and argue inductively for the resulting submatrix M'. Notice that the only columns affected by the subtraction are the columns that have a 1 in the first row. These columns, however, have a 1 in rows $1, 2, \ldots, r$ by the choice of the first column. For such a column, after the subtraction, the 1's in rows $2, \ldots, r$ become 0's, and after the first row is removed, the remaining 1's in the column will be in consecutive rows. Therefore, the matrix M' has the property of consecutive 1's in each column, and the induction goes through.

A linear programming algorithm that finds a vertex of the polytope minimizing the objective function $\sum c_j x_j$ thus solves the problem. □

Unfortunately, a linear programming algorithm is not very practical for the applications at hand. It therefore becomes interesting to seek a combinatorial algorithm that is polynomial-time. As shown in the next section, the weighted bipartite matching problem is a special case of our offline selection problem, and therefore we cannot really hope for a simple combinatorial algorithm. In practice, it might be better to use the following approximation algorithm.

THEOREM 6.2. *The offline selection problem with arbitrary costs has a* $2 \log_2 k$*-approximation polynomial-time algorithm.*

Proof. Construct a binary tree whose leaves are the minimal proper obstructions $O_1, O_2, \ldots, O_{k'}$ in left to right order, with $k' \leq k$. Place an interval at a node q if the minimal proper obstructions it contains, $O_s, O_{s+1}, \ldots, O_t$, are precisely the leaves below node q.

Unfortunately, not all intervals correspond to a single node; we argue that in general, the minimal proper obstructions covered by an interval can be decomposed into at most $2\log_2 k$ groups, with each group corresponding to a single node q. To show this, let P_i be the set of nodes on the path from the root to the leaf O_i, and consider in particular P_{s-1} and P_{t+1}, which are taken to be empty if $s = 1$ or $t = k'$, respectively. Then the nodes selected are precisely the nodes q not on P_{s-1} such that q is the right child of a node on $P_{s-1} - P_{t+1}$ plus the nodes q not on P_{t+1} such that q is the left child of node on $P_{t+1} - P_{s-1}$. If the binary tree is chosen to be balanced, the bound of $2\log_2 k$ on the number of nodes q selected follows. (By carefully choosing the binary tree, the bound can be improved by an asymptotic factor of 2.)

The algorithm then represents each interval by at most $2\log_2 k$ intervals at nodes q, each with the same cost as the original interval. This may increase the cost of an optimal solution by a factor of $2\log_2 k$. The algorithm then solves the problem with each interval at a single node q in polynomial time.

The algorithm computes at each node q, starting at the leaves O_i and moving up to the root, how to select intervals at node q so as to leave a requirement of r intervals having to be chosen at nodes higher than q, on the path from the root to q, for all possible values r, which we call the *demand* at node q. Initially, at a leaf O_i, before choosing which intervals to select at O_i, we have $r = a_i + b_i - n$. Suppose a node q has inherited demands of r' and r'' from its children and wishes to achieve demand $r \leq \max(r', r'')$ to pass on to its parent. (At a leaf, there is a single inherited demand r' as described above.) Then we must select the $\max(r', r'') - r$ intervals of least cost at q. For a given r, we carry out the calculation of total cost with the possible choices of r', r'', and select the one that gives the least total cost for the intervals chosen at q and its descendants, for the demand r under consideration. At the root, we force $r = 0$, since no demand can be satisfied at nodes higher than the root. ∎

7. Interval problems and weighted bipartite matching. We now examine the expressive power of the offline selection problem. Specifically, we define the notion of an "interval problem," show that it is equivalent to the offline median problem, and show that it includes weighted bipartite matching as a special case. It turns out that every interval problem can be expressed as a min-cost network flow problem [3]. This allows us to apply combinatorial algorithms for min-cost flow to our selection problem. We will also show that a mild generalization of the interval problem is in fact NP-hard.

DEFINITION 7.1. *Define an* interval problem *to be to minimize* $\sum c_j x_j$ *with* $x_j \in \{0, 1\}$ *subject to* k *constraints* $\sum_{S_i} x_j \geq f_i$, *with the property that each* x_j *occurs in consecutive constraints.*

We know that the offline selection problem is an interval problem. We now show the converse. The median problem will have $\delta = 1 - \frac{1}{2(k+1)}$ and k minimal proper obstructions $O_i = [i-1, i]$ for $1 \leq i \leq k$. If a variable x_j occurs in constraints $s, s+1, \ldots, t$, then we represent it by the interval $I_j = [s-1, t]$. We must also have $f_i = a_i + b_i - n = d_i - e_i$, with $a_i, b_i \geq \lceil \frac{n}{2} \rceil$. In order to have $f_i = d_i - e_i$, where d_i is the number of variables in the constraint and e_i is the number of intervals inside O_i, we put e_i single-point intervals $I_j = [p_j, p_j]$ inside O_i at $i - \frac{i}{k+1}$. To make O_i a proper obstruction, we add a zero-cost interval $I_j = O_i$. To ensure that $a_i, b_i \geq \lceil \frac{n}{2} \rceil$,

we add a sufficiently large number of zero-cost intervals $I_j = [0, k]$. It is then clear that the O_i will be the minimal proper obstructions and will give the corresponding constraint when the zero-cost intervals that were added are selected.

PROPOSITION 7.2. *Every interval problem is an offline median problem.*

We may sometimes want to force a constraint $\sum_{S_i} x_j \geq f_i$ to hold with equality. To achieve this, we choose a large $L > \sum c_j$ and add $L \sum_{S_i} x_j$ to the objective function. If the constraint can be satisfied with equality, this will be forced by the minimization, making the added term equal to Lf_i. The same L can be used to force several different constraints to hold with equality, when this is feasible.

It turns out that every interval problem can be expressed as a min-cost network flow problem and vice versa [3]. We now prove a simpler statement.

THEOREM 7.3. *The interval problem can express the weighted bipartite matching problem.*

Proof. The case in which all constraints must hold with equality can express the minimum cost bipartite perfect matching problem. To see this, consider a bipartite graph G, with k vertices on each side and costs on the edges, that has a perfect matching; we seek a minimum cost perfect matching. Let $1, 2, \ldots, k$ be the k vertices on the left, and let $1', 2', \ldots, k'$ be the k vertices on the right. The corresponding interval problem will have $2k$ constraints corresponding to these vertices, in the order $1, 2, \ldots, k, k', \ldots, 2', 1'$. An edge from i to j' will correspond to a variable occurring in the constraints from i to j' in this order. To force exactly one of the edges incident to vertex 1 to be selected, we use the bound $f_1 = 1$ for the first constraint, with equality. To force exactly one additional edge incident to vertex 2 to be selected, we use the bound $f_2 = 2$ for the second constraint, with equality. In general, we have $f_i = f_{i'} = i$, with equality. This completes the representation. □

Consequently, the weighted bipartite matching problem is a special case of the weighted offline median problem. We use this expressiveness of the weighted offline median problem to show that a slight generalization is computationally hard.

The *median problem with multiplicities* is the median problem with the additional provision that some specified intervals, when queried, return two points instead of just one. This is like having, in the median problem, pairs of identical intervals that must be simultaneously queried or not queried.

THEOREM 7.4. *The weighted offline median problem with multiplicities is NP-complete.*

Proof. The problem corresponds as before to a general interval problem, except that now, for some of the variables, the quantity $2x_j$ instead of x_j appears in the constraints. We can again force equality to hold in the constraints by using appropriate costs. The special case of bipartite matching from before now generalizes to a bipartite flow problem. In this problem, vertices on the left have a given supply amount, vertices on the right have a given demand amount, and vertices on the left are joined to vertices on the right by edges of capacity 1 or 2. The supplies and demands must be satisfied by sending flow across the edges but in such a way that the flow across an edge is either zero or the full capacity of the edge.

We show that this bipartite flow problem is NP-complete. The reduction is from 3SAT. In fact, we assume that every variable x in the 3SAT instance occurs at most twice as a positive literal x and at most twice as a negative literal $\overline{x}$. Every 3SAT instance can be made to satisfy this assumption by replacing every variable with several variables forced to be equal by a cycle of implications; the implications in the cycle account for one positive and one negative occurrence, so each variable in the

cycle can be used again, once as positive and once as negative.

To express such a 3SAT instance as a bipartite flow problem, we put a vertex on the right for each literal x or $\overline{x}$, with demand 2. We put a vertex on the left for each clause, with supply 1, and with three capacity 1 edges joining it to the literals occurring in the clause on the right. Thus the supply 1 must be sent to one of the three literals occurring in the clause, which we can take to be a satisfying literal for the clause. Notice that the demand bound of 2 will hold, since no literal occurs in more than two clauses. We must ensure that complementary literals are not both chosen as satisfying literals. To achieve this, we put a vertex on the left for each variable x, with supply 2, and with capacity 2 edges joining it to both literals x and $\overline{x}$. Thus only one of the two literals will have demand left to be chosen by the clauses on the left. Thus the supply on the left can be sent while satisfying the demand bounds on the right if and only if the instance has a solution. It then remains to force the demands of 2 to be met exactly, and this is done by providing additional $2v - w$ units of supply to be sent along capacity 1 edges to any vertex on the right, where v is the number of variables and w is the number of clauses. This completes the proof of NP-completeness. □

It is natural to consider the related problem, where, when an interval I_j is queried, instead of returning a single point p_j in I_j, we obtain a subinterval of I_j of length at most δ'. If the new parameter δ' satisfies $\delta' \leq \delta$, where δ is the parameter for the required precision as before, then all the results obtained above go through (unit-cost or arbitrary cost, online or offline), since an obstruction O has length greater than δ, so that an interval containing it will no longer contain it after the interval length is reduced to at most δ'. On the other hand, if $\delta' > \delta$, then the problem has no solution (unless no queries are needed), since queried intervals I_j can be answered so as to still contain a sufficiently small obstruction O.

Acknowledgment. We would like to thank Suresh Venkatasubramanian for several helpful discussions.

REFERENCES

[1] T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom, *Computing the median with uncertainty*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 602–607.

[2] O. H. Ibarra and C. E. Kim, *Fast approximation algorithms for the knapsack and sum of subset problems*, J. ACM, 22 (1975), pp. 463–468.

[3] G. L. Nemhauser and L. A. Wosley, *Integer and Combinatorial Optimization*, Wiley, New York, 1988.

[4] C. Olston and J. Widom, *Offering a precision-performance tradeoff for aggregation queries over replicated data*, in Proceedings of the 26th International Conference on Very Large Data Bases, Cairo, Egypt, 2000, http://www-db.stanford.edu/pub/papers/trapp-ag.ps.

SIAM J. COMPUT.
Vol. 32, No. 2, pp. 548–556

ACCELERATION OF EUCLIDEAN ALGORITHM AND RATIONAL NUMBER RECONSTRUCTION*

XINMAO WANG† AND VICTOR Y. PAN‡

Abstract. We accelerate the known algorithms for computing a selected entry of the extended Euclidean algorithm for integers and, consequently, for the modular and numerical rational number reconstruction problems. The acceleration is from quadratic to nearly linear time, matching the known complexity bound for the integer gcd, which our algorithm computes as a special case.

Key words. extended Euclidean algorithm, rational number reconstruction

AMS subject classifications. 68W40, 68W30, 68Q25

PII. S0097539702408636

1. Introduction.

A customary approach in computer algebra is to perform computations with rational numbers modulo a large integer q (a prime, prime power, or product of several selected primes) and then to reconstruct the rational output from its value modulo q [GG99]. In particular, the *modular rational number reconstruction* is the final stage of the solution of a nonsingular linear system of n equations by means of p-adic lifting [MC79], [D82], [P02] (see [GG99], [S86], [UP83], [Z93], [HW60] for other important applications).

PROBLEM 1.1 (modular rational number reconstruction). *Compute a pair of integers (η, δ) from three positive integers m, n, k such that*

$$|\eta| < k < m, \quad 1 \le \delta \le m/k, \quad \eta = n\delta \bmod m. \tag{1.1}$$

PROBLEM 1.1a. *Compute all coprime solutions (η, δ) to Problem* 1.1.

There always exists a solution to Problem 1.1. There are at most two solutions to Problem 1.1a, and at most one of them satisfies $|\eta| < k/2$ [GG99, Theorem 5.26]. To ensure unique correct reconstruction of η and δ, having some upper bounds on $|\eta|$ and δ (e.g., Hadamard's bound applies to the coordinates of the rational solution to a linear system of equations), we may double the available bound k on $|\eta|$, compute one solution to (1.1), and either output it if $|\eta| < k/2$ or otherwise compute and output the other solution.

A related problem of *numerical rational number reconstruction* or *rational roundoff* is the problem of computing the best rational approximation s/t to a given rational n/m such that $1 \le t \le k$.

PROBLEM 1.2 (rational roundoff). *Compute all rational numbers s/t from three positive integers m, n, k such that*

$$1 \le t \le k, \quad |s/t - n/m| \text{ is minimal.} \tag{1.2}$$

*Received by the editors June 3, 2002; accepted for publication (in revised form) October 17, 2002; published electronically February 20, 2003. Some results of this paper were presented at ISSAC 2002; we are grateful to the ACM for the permission to reuse them.

http://www.siam.org/journals/sicomp/32-2/40863.html

†Ph.D. Program in Mathematics, Graduate School of CUNY, New York, NY 10016 (xwang2@gc.cuny.edu).

‡Department of Mathematics and Computer Science, Lehman College of CUNY, Bronx, NY 10468 (vpan@lehman.cuny.edu). This author's research was supported by NSF grant CCR 9732206 and PSC CUNY award 66383-0032.

Problem 1.2 is closely related to computing *Diophantine approximations* to a real number [HW60], [H82], [GG99] and extends the following problem.

PROBLEM 1.2a (see [UP83]). *Given a rational number $\alpha = m/n$ and a natural number k, find a rational number p/q such that $1 \le q \le k$ and $|\alpha - p/q| < 1/(2k^2)$.*

Problem 1.2a may have no solution, but the solution is unique if it exists. In section 5, we show that the solution to Problem 1.2 is also unique.

Dirichlet [D1842] showed that, for any real numbers α and $0 < \epsilon \le 1$, there exist integers p and q such that $|\alpha - p/q| < \epsilon/q$ and $1 \le q \le \epsilon^{-1}$. In particular, let p_i/q_i be the ith *convergent* of α (i.e., the ith term in the continued fraction approximation for α); then $|\alpha - p_i/q_i| \le 1/(q_i q_{i+1}) < 1/q_i^2$ [HW60], [H82]. Furthermore, Hurwitz [H1891] showed that at least one of the two consecutive convergents of α satisfies $|\alpha - p/q| < 1/(2q^2)$, and at least one of the three consecutive convergents of α satisfies $|\alpha - p/q| < 1/(\sqrt{5}q^2)$. On the other hand, Legendre [L1798] showed that if $|\alpha - p/q| < 1/(2q^2)$, then p/q is a convergent of α. Therefore, Problems 1.2 and 1.2a are reduced to computing the convergents of α.

The common approach to the solution of the problems of modular and numerical rational number reconstruction is by applying the extended Euclidean algorithm to m and n [HW60]. Hereafter, we refer to this algorithm as the *EEA* and we seek faster solution algorithms based on accelerating the EEA. The algorithm produces a sequence of triples (r_j, s_j, t_j), $j = 1, \dots, l$ (notation used in [GG99]; see our Remark 2.10). In our case, we need only the triples $(r_{j-1}, s_{j-1}, t_{j-1})$ and (r_j, s_j, t_j) for a specially selected j. Extension from computing these triples to the solution to Problems 1.1 and 1.1a is shown in full detail in [GG99, Theorem 5.26]. We show an alternative approach, which is more directly related to our modification of the EEA. We also extend the known reduction of the Diophantine approximation to the EEA to solve Problem 1.2. Our main result, however, is the acceleration of the EEA and consequently the solution of all the listed problems. The known algorithms compute the desired pair of the EEA triples and thus solve Problems 1.1, 1.1a, 1.2, and 1.2a by using

$$f(d) = O(d^2)$$

bit operations, where $d = \lfloor \log_2 m \rfloor$, $m \ge n$. We speed up the computation by the factor of almost d; that is, we decrease the above bit cost bound to the level

$$\rho(d) = O(\mu(d) \log d), \tag{1.3}$$

provided that $\mu(d)$ bit operations are sufficient to multiply two integers modulo $2^d + 1$, and (see [SS71]) we have

$$\mu(d) = O((d \log d) \log \log d). \tag{1.4}$$

A similar acceleration is known for the Euclidean algorithm applied to polynomials [M73], [AHU74], [BGY80], but in the integer case a well-known additional difficulty is due to the carries. Among the known methods, only the Knuth–Schönhage algorithm [S71] has settled the problem for integers but only in the special case in which $j = l$ and the triple (r_l, s_l, t_l) terminates the Euclidean algorithm, that is, where r_l is the gcd. In our work, we were motivated by the following excerpt from [GG99, p. 305] on the EEA for integers:

> *The method also works for integers, although there are some complications due to the carries,*

and by the recent comments of expert Joachim von zur Gathen on the state of the art which he sent by email to one of the present authors:

> *Yes, I suppose rational number reconstruction can be done in time* $O(m(n)\log n)$ *for* n*-bit numbers and a given upper bound on the denominator. This is alluded to in* [GG99]*, as you observed. But we do not give a proof, and I do not know any rigorous proof in the literature. I can imagine roughly what needs to be done, but it will be quite messy.*

In the next sections, we clear the cited mess and come out with a desired algorithm, which solves the gcd problem as a special case (see Remark 4.3(ii)). Our construction relies on computing a matrix sequence $\{Q_i,\ i = 0, 1, \dots\}$, which represents the quotients and cofactors computed in the EEA, rather than on computing just the remainder sequence $\{r_i,\ i = 0, 1, \dots\}$. This enables a simpler control over the growth of the magnitude of the entries of the Q_i than we would have had over the decrease of the r_i.

We organize our paper as follows. After some preliminaries in the next section, we prove our technical results on the EEA in section 3. In section 4, we present our main algorithm. In section 5, we apply it to accelerate the modular and numerical rational number reconstruction. Our proof of our main result is substantially simpler than in the proceedings version [PW02].

2. Some basic results. Hereafter, we write log to replace $\log_2$ unless specified otherwise.

DEFINITION 2.1. *$\mathbb{Z}$ is the ring of integers. $\lfloor x\rfloor$ and $\lceil x\rceil$ are two integers closest to a real number x such that $\lfloor x\rfloor \le x \le \lceil x\rceil$. $\{x\} = x - \lfloor x\rfloor$. $|A| = \max_{i,j} |a_{i,j}|$ for any real matrix $A = (a_{i,j})_{i,j}$. $m \bmod n$ is defined to be $m - n\lfloor m/n\rfloor$ for $m, n \in \mathbb{Z}$, and $n > 0$.*

ALGORITHM 2.2 (Euclidean algorithm).

INPUT: *A pair of natural numbers (m, n), $m \ge n$.*

OUTPUT: $\gcd(m, n)$.

COMPUTATION: *Write $r_0 = m$, $r_1 = n$. Compute*

$$r_{i+1} = r_{i-1} \bmod r_i$$

for $i = 1, 2, \dots, l$ until $r_{l+1} = 0$. Output r_l.

DEFINITION 2.3. *Let $\binom{r_{i-1}}{r_i} = P_i \binom{r_i}{r_{i+1}}$, where*

$$P_i = \begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix}, \quad q_i = \lfloor r_{i-1}/r_i\rfloor, \quad i = 1, 2, \dots, l, \tag{2.1}$$

$$Q_i = \begin{pmatrix} a_i & b_i \\ c_i & d_i \end{pmatrix} = P_1 P_2 \cdots P_i, \quad i = 1, 2, \dots, l, \tag{2.2}$$

$$Q_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad Q_{l+1} = \begin{pmatrix} \infty & \infty \\ \infty & \infty \end{pmatrix}.$$

The sequence $\{r_i\}_{i=0}^{l}$ is called the remainder sequence*, and the sequence $\{Q_i\}_{i=0}^{l}$ is called the* matrix sequence*. The* extended Euclidean algorithm (EEA) *outputs both sequences $\{r_i\}_{i=0}^{l}$ and $\{Q_i\}_{i=0}^{l}$ (see Remark* 2.10*).*

For a given pair (m, n) and the sequence $\{Q_i\}$, we can immediately compute the sequence $\{r_i\}$ because

$$\det P_i = -1, \quad \det Q_i = (-1)^i, \tag{2.3}$$

$$\begin{pmatrix} m \\ n \end{pmatrix} = Q_i \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}, \quad Q_i^{-1} = (-1)^i \begin{pmatrix} d_i & -b_i \\ -c_i & a_i \end{pmatrix} \tag{2.4}$$

for all $i = 1, 2, \dots, l$.

Our main task is to solve the following problem.

PROBLEM 2.4 (selected output of the EEA).

INPUT: *Integers m, n, h such that $m \geq n \geq 1$, $h \geq 0$.*

OUTPUT: *The unique Q_i such that $|Q_i| \leq 2^h < |Q_{i+1}|$.*

In the remaining part of this section, we state some simple auxiliary properties of the remainders r_i and the matrices Q_i.

THEOREM 2.5. *$r_i > r_{i+1} > 0$, $r_i \geq r_{i+1} + r_{i+2}$ for $i = 0, 1, \dots, l-1$.*

THEOREM 2.6 (cf. Definition 2.3 for a_i, b_i, c_i, q_i).

(i) *$b_i = a_{i-1}$, $d_i = c_{i-1}$ for $i = 1, 2, \dots, l$.*

(ii) *$a_i = a_{i-1}q_i + a_{i-2} > a_{i-1}$, $c_i = c_{i-1}q_i + c_{i-2} > c_{i-1}$ for $i = 2, 3, \dots, l$.*

(iii) *$a_{i-2} = a_i \bmod a_{i-1}$, $c_{i-2} = c_i \bmod c_{i-1}$ for $i = 3, 4, \dots, l$.*

(iv) *$a_0 > c_0$, $a_1 \geq c_1$, $a_i > c_i$ for $i = 2, 3, \dots, l$.*

COROLLARY 2.7. *Q_{i-1} can be computed from Q_i by Theorem* 2.6 (i), (iii).

COROLLARY 2.8.

(i) *$|Q_i| = a_i$ for $i = 0, 1, \dots, l$.*

(ii) *$|Q_i| \geq |Q_{i-1}| + |Q_{i-2}|$ for $i = 2, 3, \dots, l$.*

COROLLARY 2.9. *$m/2 < r_i|Q_i| \leq m$ for $i = 0, 1, \dots, l$.*

Remark 2.10. Note an equivalent customary representation of the EEA's output by the sequences $\{r_i\}$, $\{s_i\}$, $\{t_i\}$ (with the notation in [GG99]), where $s_i = (-1)^i d_i$, $t_i = (-1)^{i-1} b_i$.

Remark 2.11. By Corollary 2.9, we have $|Q_i| \leq m$ for $i \leq l$, so it is sufficient to consider Problem 2.4 for $h \leq d + 1$, $d = \lfloor \log m \rfloor$.

Remark 2.12. The remainder r_i defined by (2.4) for the solution Q_i of Problem 2.4 equals the gcd of m and n if and only if $Q_{i+1} = \left(\begin{smallmatrix} \infty & \infty \\ \infty & \infty \end{smallmatrix}\right)$, which is always the case for $h = d + 1$.

3. The EEA for a modified input. To accelerate the solution of Problem 2.4, we apply the divide-and-conquer techniques. Roughly, the idea is to solve Problem 2.4 in two steps. In each step, Problem 2.4 is solved for h replaced by $\lfloor h/2 \rfloor$, and the output of the first step is used as the input of the second step. We are going to show that

(i) this leads to the same desired output, and

(ii) the computational cost of the reduction to the pair of half-size problems is small.

A basic observation is that the matrix sequence $\{Q_i\}$ depends only on the quotient m/n. That is, for another input values m^* and n^* such that $m^*/n^* = m/n$, the Euclidean algorithm computes the same matrices $Q_i^* = Q_i$ for all i. A relatively small perturbation of the quotient m/n should not affect the first several terms of the sequence $\{Q_i\}$, using which is enough to solve the problem for smaller h. That is, we may replace m and n by smaller integers m^* and n^* provided that $m^*/n^* \approx m/n$. For the input values m^* and n^*, we denote by $\{r_i^*\}$ the remainder sequence and by $\{Q_i^*\}$ the matrix sequence. Next, we specify some bounds on the allowed perturbations of m/n for which $Q_i = Q_i^*$ and then state our main theorem.

THEOREM 3.1. *Suppose* $m^* = \lfloor m/\lambda \rfloor$ *and* $n^* = \lfloor n/\lambda \rfloor$ *for a positive integer* λ. *For any given integer* i, *if*

$$r^*_{i+2} \geq |Q^*_{i+1}| \quad or \quad r_{i+2} \geq \lambda |Q_{i+1}|,$$

then $Q_i = Q^*_i$.

Proof. (i) Suppose $r^*_{i+2} \geq |Q^*_{i+1}|$. Write $\binom{u_j}{v_j} = Q^{*-1}_j \binom{m}{n}$ for $j = 0, 1, \ldots, i+1$. Then we have

$$\begin{pmatrix} u_{j+1} \\ v_{j+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q^*_{j+1} \end{pmatrix} \begin{pmatrix} u_j \\ v_j \end{pmatrix}.$$

Therefore, $u_{j+1} = v_j$ for $j = 0, 1, \ldots, i$. Furthermore, extending (2.4) to (m^*, n^*), we obtain that

$$\begin{pmatrix} r^*_j \\ r^*_{j+1} \end{pmatrix} = Q^{*-1}_j \begin{pmatrix} m^* \\ n^* \end{pmatrix},$$

$$\begin{pmatrix} u_j \\ v_j \end{pmatrix} = \begin{pmatrix} r^*_j \\ r^*_{j+1} \end{pmatrix} \lambda + Q^{*-1}_j \begin{pmatrix} m - m^*\lambda \\ n - n^*\lambda \end{pmatrix}.$$

By (2.4) we also know that, in each row of Q^{*-1}_j, one of the entries is nonnegative, and another is nonpositive, and their absolute values are bounded by $|Q^*_{j-1}|$ in the first row and by $|Q^*_j|$ in the second row. Therefore, we have

$$v_j > (r^*_{j+1} - |Q^*_j|)\lambda$$

and

$$u_j - v_j > (r^*_j - |Q^*_{j-1}|)\lambda - (r^*_{j+1} + |Q^*_j|)\lambda \geq (r^*_{j+2} - |Q^*_{j+1}|)\lambda.$$

So, by assumption, $u_j > v_j > 0$ for $j = 1, 2, \ldots, i$. Now we have $u_0 = m$, $u_1 = n$, $u_{j+1} = u_{j-1} \bmod u_j$ for $j = 1, 2, \ldots, i$. So $u_j = r_j$ and $Q_j = Q^*_j$ for $j = 0, 1, \ldots, i$.

(ii) Suppose $r_{i+2} \geq \lambda |Q_{i+1}|$. Write $\binom{x_j}{y_j} = Q^{-1}_j \binom{m^*}{n^*}$ for $j = 0, 1, \ldots, i+1$. Then we have

$$\begin{pmatrix} x_{j+1} \\ y_{j+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q_{j+1} \end{pmatrix} \begin{pmatrix} x_j \\ y_j \end{pmatrix}.$$

Therefore, $x_{j+1} = y_j$ for $j = 0, 1, \ldots, i$. Furthermore, by (2.4), we extend the above expression for x_j and y_j as follows:

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} = \begin{pmatrix} r_j \\ r_{j+1} \end{pmatrix} \lambda^{-1} - Q^{-1}_j \begin{pmatrix} m/\lambda - m^* \\ n/\lambda - n^* \end{pmatrix}.$$

Now, similarly as in part (i), we deduce that $y_j > r_{j+1}\lambda^{-1} - |Q_j|$ and $x_j - y_j > (r_j\lambda^{-1} - |Q_{j-1}|) - (r_{j+1}\lambda^{-1} + |Q_j|) \geq (r_{j+2}\lambda^{-1} - |Q_{j+1}|)$. So, by assumption, $x_j > y_j > 0$ for $j = 1, 2, \ldots, i$. Now we have $x_0 = m^*$, $x_1 = n^*$, $x_{j+1} = x_{j-1} \bmod x_j$ for $j = 1, 2, \ldots, i$. So $x_j = r^*_j$ and $Q_j = Q^*_j$ for $j = 0, 1, \ldots, i$. □

COROLLARY 3.2. *Suppose* $m^* = \lfloor m/\lambda \rfloor$, $n^* = \lfloor n/\lambda \rfloor$ *for a positive integer* λ. *For any given integer* i, *if*

$$m^* \geq 2|Q^*_{i+2}| \cdot |Q^*_{i+1}| \quad or \quad m \geq 2\lambda |Q_{i+2}| \cdot |Q_{i+1}|,$$

then $Q_i = Q_i^*$.

Proof. Combine the assumed bound on m^* and m with the first inequality of Corollary 2.9 extended also to m^*, r_i^*, Q_i^*, and arrive at the bounds on r_{i+2}^* and r_{i+2} in Theorem 3.1. ∎

THEOREM 3.3 (main theorem). *Suppose* $m^* = \lfloor m/\lambda \rfloor$, $n^* = \lfloor n/\lambda \rfloor$ *for a positive integer* λ, *and* K *is a given positive integer such that* $m^* \geq 2K^2$. *If* $|Q_i^*| \leq K < |Q_{i+1}^*|$, *then* $Q_j = Q_j^*$ *for all* $j \leq i-2$ *and* $|Q_j| \leq K < |Q_{j+1}|$ *for some* j *such that* $i-2 \leq j \leq i+2$.

Proof. By Corollary 3.2, we have $Q_j = Q_j^*$ for $j \leq i-2$. If $|Q_{i+3}| > K$, then we are done. Otherwise, we have $m \geq \lambda m^* \geq 2\lambda K^2 \geq 2\lambda |Q_{i+3}|^2$. By applying Corollary 3.2 again, we obtain $Q_{i+1} = Q_{i+1}^*$, $Q_i = Q_i$. ∎

4. Our main algorithm.

ALGORITHM 4.1 (selected output of the EEA).

INPUT: *A triple of integers* (m, n, h) *such that* $m \geq n > 0, h \geq 0$.

OUTPUT: *The unique matrix* Q_k *such that* $|Q_k| \leq 2^h < |Q_{k+1}|$.

COMPUTATION: *Let* $d = \lfloor \log m \rfloor$.

1. *When* $h \leq \lfloor d/2 \rfloor - 1$, *let* $\lambda = 2^{d-2h-1}$, $m^* = \lfloor m/\lambda \rfloor$, *and* $n^* = \lfloor n/\lambda \rfloor$; *then* $2^{2h+1} \leq m^* \leq m/2$. *We first apply the algorithm to the input* (m^*, n^*, h) *and have the output* Q_i^*. *Theorem* 3.3 *for* $K = 2^h$ *implies that* $Q_{i-2} = Q_{i-2}^*$ *and* $|Q_k| \leq 2^h < |Q_{k+1}|$ *for some* $i-2 \leq k \leq i+2$. *We may compute* $Q_{i-2} = Q_{i-2}^*$ *from* Q_i^* *(cf. Corollary* 2.7*) and then find* Q_k *in a few Euclidean steps.*
2. *When* $\lfloor d/2 \rfloor \leq h \leq d-1$, *we first apply the algorithm to find* $|Q_i| \leq 2^{\lfloor h/2 \rfloor} < |Q_{i+1}|$. *Next we apply the algorithm again for the input* $(r_i, r_{i+1}, \lfloor h/2 \rfloor)$ *and have the output* $\tilde{Q}_j$. *Now we have* $Q_{i+j} = Q_i \tilde{Q}_j$, $|Q_{i+j}| < 2^{h+1}$, *and* $|Q_{i+j+2}| > 2^{h-1}$. *Then* $|Q_k| \leq 2^h < |Q_{k+1}|$ *for some* $i+j-2 \leq k \leq i+j+2$, *and we may find* Q_k *in a few Euclidean steps.*
3. *When* $h \geq d$, *we first apply the algorithm to find* $|Q_i| \leq 2^{d-1} < |Q_{i+1}|$. *Then* $|Q_k| \leq 2^h < |Q_{k+1}|$ *for some* $i \leq k \leq i+4$, *and we may find* Q_k *in a few Euclidean steps.*

THEOREM 4.2. *Let* $f(d, h)$ *be the bit cost of performing Algorithm* 4.1 *for the input* (m, n, h), *where* $d = \lfloor \log m \rfloor$. *Then we have*

$$f(d, h) = O(\mu(d) \log h)$$

for μ *in* (1.4).

Proof. By inspection of the algorithm, we have

$$f(d,h) = \begin{cases} f(2h+1, h) + O(\mu(d)) & \text{if } h \leq \lfloor \frac{d}{2} \rfloor - 1, \\ f(d, \lfloor \frac{h}{2} \rfloor) + f(d - \lfloor \frac{h}{2} \rfloor, \lfloor \frac{h}{2} \rfloor) + O(\mu(d)) & \\ & \text{if } \lfloor \frac{d}{2} \rfloor \leq h \leq d-1, \\ f(d, d-1) + O(\mu(d)) & \text{if } h \geq d. \end{cases}$$

Let us write $F(h) = f(2h+1, h)$. Then

$$F(h) = 2F(\lfloor h/2 \rfloor) + O(\mu(2h)),$$

and we obtain that

$$F(h) = O(\mu(2h) \log h).$$

By recursively combining this bound with the above expressions for $f(d,h)$, we obtain

$$f(d,h)=\sum_{i=1}^{1+\lfloor \log h\rfloor}(F(\lfloor h/2^i\rfloor)+O(\mu(d)))=O(\mu(d)\log h). \qquad \square$$

Remark 4.3.

(i) We may easily extend Algorithm 4.1 to compute the matrix Q_i (at the bit cost $O(\mu(d)\log\log K)$), such that $|Q_i|\le K<|Q_{i+1}|$ for any real $K\ge 1$, not just for $K=2^h$.

(ii) Due to Remark 2.12, we may also easily extend Algorithm 4.1 to find the remainder r_i (at the bit cost $O(\mu(d)\log\log(m/K))$, such that $r_i\ge K>r_{i+1}$ for any real $1\le K\le m$. By choosing $K=1$, we compute $r_i=\gcd(m,n)$.

5. Applications to rational number reconstruction. Let us next extend Algorithm 4.1 to solve Problems 1.1, 1.1a, 1.2, and 1.2a of rational number reconstruction.

Solution of Problems 1.1 *and* 1.1a. Note that (cf. (2.4))

$$\begin{pmatrix} r_i\\ r_{i+1}\end{pmatrix}=Q_i^{-1}\begin{pmatrix} m\\ n\end{pmatrix}=(-1)^i\begin{pmatrix} d_i & -b_i\\ -c_i & a_i\end{pmatrix}\begin{pmatrix} m\\ n\end{pmatrix}.$$

Therefore,

$$(-1)^i r_{i+1}=na_i \bmod m \text{ for all } i.$$

Let i be such that $a_i\le m/k<a_{i+1}$.

1. Since $a_i\le m/k$ and $r_{i+1}\le \frac{m}{a_{i+1}}<k$, we obtain a solution $\big((-1)^i r_{i+1},\ a_i\big)$ to Problem 1.1.
2. Suppose Problem 1.1a has a solution (η,δ) such that $\eta/\delta=(-1)^i r_{i+1}/a_i$; then $(\eta,\delta)=\big((-1)^i r_{i+1},\ a_i\big)$ and $\gcd(r_{i+1},a_i)=1$. Indeed, a_i divides δ because $\frac{n\delta-\eta}{m}=\frac{c_i\delta}{a_i}\in\mathbb{Z}$ and $\gcd(a_i,c_i)=1$.
3. Suppose Problem 1.1a has a solution such that $\eta/\delta\ne(-1)^i r_{i+1}/a_i$. Then the solution is unique. (Indeed, it follows from $a_i\eta-(-1)^i r_{i+1}\delta=0 \bmod m$ that $a_i(-1)^{i-1}\eta+r_{i+1}\delta=m$ and $(-1)^{i-1}\eta\ge 0$. Furthermore, if there are two such solutions (η_1,δ_1) and (η_2,δ_2), then $\eta_1\delta_2-\eta_2\delta_1=0 \bmod m$. So $\eta_1\delta_2-\delta_1\eta_2$ equals $0, m$, or $-m$. Combine $(-1)^{i-1}\eta_1\ge 0$ and $(-1)^{i-1}\eta_2\ge 0$ to deduce that only $\eta_1\delta_2-\eta_2\delta_1=0$ can hold.) Since $m=a_i r_i+r_{i+1}a_{i-1}$ by (2.4), we have $((-1)^{i-1}\eta-r_i)a_i=(a_{i-1}-\delta)r_{i+1}$. Therefore, $(\eta,\delta)=\big((-1)^{i-1}(r_i-tr_{i+1}),\ a_{i-1}+ta_i\big)$ for a real t. Note that $a_{i-1}+ta_i\in\mathbb{Z}$, $\frac{n\delta-\eta}{m}=c_{i-1}+tc_i\in\mathbb{Z}$, and $\gcd(a_i,c_i)=1$, and so $t\in\mathbb{Z}$. If $r_i<k$, then $(\eta,\delta)=\big((-1)^{i-1}r_i,\ a_{i-1}\big)$ defines the unique solution. If $r_i\ge k$, then by applying the inequalities $|\eta|<k$ and $\delta\le m/k$, we obtain $\frac{r_i-k}{r_{i+1}}<t\le\frac{m/k-a_{i-1}}{a_i}$. Therefore, the unique solution must be defined by the unique integer t in the interval $\big(\frac{r_i-k}{r_{i+1}},\frac{m/k-a_{i-1}}{a_i}\big]$. $\square$

COROLLARY 5.1. *Problems* 1.1 *and* 1.1a *of modular rational number reconstruction can be solved by using* $\rho(d)$ *bit operations for* ρ *in* (1.3).

Solution of Problems 1.2 *and* 1.2a. Recall that c_i/a_i is the ith continued fraction approximation of n/m, and $|\frac{c_i}{a_i}-\frac{n}{m}|<|\frac{s}{t}-\frac{n}{m}|$ for all i,s,t, where $1\le t<a_i$ (see [HW60, Theorem 181]). In particular, $|\frac{c_i}{a_i}-\frac{n}{m}|<|\frac{c_{i-1}}{a_{i-1}}-\frac{n}{m}|$ for all i. Let i be such that $a_i\le k<a_{i+1}$. Suppose

$$|\tfrac{s}{t}-\tfrac{n}{m}|\le|\tfrac{c_i}{a_i}-\tfrac{n}{m}|,\quad a_i<t\le k.$$

Then

$$\begin{aligned} & |\tfrac{c_i}{a_i} - \tfrac{s}{t}| \le 2|\tfrac{c_i}{a_i} - \tfrac{n}{m}| = \tfrac{2r_{i+1}}{a_i m} < \tfrac{2}{a_i k} \\ \Longrightarrow \quad & |c_i t - a_i s| < \tfrac{2t}{k} \le 2 \\ \Longrightarrow \quad & |c_i t - a_i s| = 1 = |c_i a_{i-1} - a_i c_{i-1}| \\ \Longrightarrow \quad & (s,t) = (c_i,\ a_i)\tau \pm (c_{i-1},\ a_{i-1}) \end{aligned}$$

for a real τ. Since $t > a_i > a_{i-1}$ and $tc_{i-1} - sa_{i-1} = (a_i c_{i-1} - c_i a_{i-1})\tau = (-1)^i\tau$, τ is a positive integer. Furthermore, observe that $\frac{c_i}{a_i} - \frac{n}{m} = \frac{(-1)^{i+1} r_{i+1}}{a_i m}$ and $\frac{c_{i-1}}{a_{i-1}} - \frac{n}{m} = \frac{(-1)^i r_i}{a_{i-1} m}$ have opposite signs, and recall that $|\frac{s}{t} - \frac{n}{m}| \le |\frac{c_i}{a_i} - \frac{n}{m}| < |\frac{c_{i-1}}{a_{i-1}} - \frac{n}{m}|$, so $(s,t) = \tau(c_i,\ a_i) + (c_{i-1},\ a_{i-1})$. Therefore, (s,t) is the unique solution to Problem 1.2 for $\tau = \lfloor \frac{k - a_{i-1}}{a_i} \rfloor$. □

COROLLARY 5.2. *Problems* 1.2 *and* 1.2a *of rational roundoff can be solved by using* $\rho(d)$ *bit operations for* ρ *in* (1.3).

Acknowledgment. We are grateful to Joachim von zur Gathen for his expert advice on the state of the art of the bit complexity of rational number reconstruction and for his helpful comments on the original draft of our paper.

REFERENCES

[AHU74] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.

[BGY80] R. P. BRENT, F. G. GUSTAVSON, AND D. Y. Y. YUN, *Fast solution of Toeplitz systems of equations and computation of Padé approximations*, J. Algorithms, 1 (1980), pp. 259–295.

[D1842] G. LEJEUNE DIRICHLET, *Verallgemeinerung eines Satzes aus der Lehre von den Kettenbrüchen nebst einigen Anwendungen auf die Theorie der Zahlen*, in Bericht über die zur Bekanntmachung geeigneten Verhandlungen der Königlich Preussischen Akademie der Wissenschaften zu Berlin, 1842, pp. 93–95 (reprinted in *G. Lejeune Dirichlet's Werke, Vol.* I, L. Kronecher, ed., G. Reimer, Berlin, 1889, pp. 635–638 (reprinted: Chelsea, New York, 1969)).

[D82] J. D. DIXON, *Exact solution of linear equations using p-adic expansions*, Numer. Math., 40 (1982), pp. 137–141.

[GG99] J. VON ZUR GATHEN AND J. GERHARD, *Modern Computer Algebra*, Cambridge University Press, Cambridge, UK, 1999.

[H82] L. K. HUA, *Introduction to Number Theory*, Springer-Verlag, Berlin, 1982.

[H1891] A. HURWITZ, *Ueber die angenäherte Darstellung der Irrationalzahlen durch rationale Brüche*, Math. Ann., 39 (1891), pp. 279–284 (reprinted in Mathematische Werke von Adolf Hurwitz, Zweiter Band, Birkhäuser, Basel, 1963, pp. 122–128).

[HW60] G. H. HARDY AND E. M. WRIGHT, *An Introduction to the Theory of Numbers*, 4th ed., Clarendon Press, Oxford, UK, 1960.

[L1798] A. M. LEGENDRE, *Essai sur la théorie des nombres*, J. B. M. Duprat, Paris, 1798 (4th edition reprinted: *Théorie des nombres*, A. Blanchard, Paris, 1979).

[M73] R. MOENCK, *Fast computation of GCDs*, in Proceedings of 5th ACM Annual Symposium on Theory of Computing, ACM, New York, 1973, pp. 142–171.

[MC79] R. T. MOENCK AND J. H. CARTER, *Approximate algorithms to derive exact solutions to systems of linear equations*, in Proceedings of EUROSAM, Lecture Notes in Comput. Sci. 72, Springer-Verlag, Berlin, 1979, pp. 63–73.

[P02] V. Y. PAN, *Can we optimize Toeplitz/Hankel computations?*, in Proceedings of the Fifth International Workshop on Computer Algebra in Scientific Computing (CASC 2002, Yalta, Ukraine), V. G. Ganzha, E. W. Mayr, and E. V. Vorozhzov, eds., Institut fuer Informatik, Technische Universitaet Muenchen, Garching, Germany, 2002, pp. 253–264.

[PW02] V. Y. PAN AND X. WANG, *Acceleration of Euclidean algorithm and extensions*, in Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, T. Mora, ed., ACM, New York, 2002, pp. 207–213.

[S71] A. Schönhage, *Schnelle Berechnung von Kettenbruchentwicklungen*, Acta Inform., 1 (1971), pp. 139–144.
[S86] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley, New York, 1986.
[SS71] A. Schönhage and V. Strassen, *Schnelle Multiplikation grosse Zahlen*, Computing, 7 (1971), pp. 281–292.
[UP83] S. Ursic and C. Patarra, *Exact solution of systems of linear equations with iterative methods*, SIAM J. Algebraic Discrete Methods, 4 (1983), pp. 111–115.
[Z93] R. E. Zippel, *Effective Polynomial Computation*, Kluwer Academic Publishers, Norwell, MA, 1993.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 557–585

REARRANGEABILITY OF $(2n-1)$-STAGE SHUFFLE-EXCHANGE NETWORKS*

HASAN ÇAM†

Abstract. Rearrangeable networks can realize each and every permutation in one pass through the network. Shuffle-exchange networks provide an efficient interconnection scheme for implementing various types of parallel processes. Whether $(2n-1)$-stage shuffle-exchange networks with $N=2^n$ inputs/outputs are rearrangeable has remained an open question for approximately three decades. This question has been answered affirmatively in this paper. An important corollary of the main result is the proof that two passes through an Omega network are sufficient and necessary to implement any permutation. In obtaining the main results of this paper, *frames* that look like grids with horizontal links of different lengths are shown to be remarkable tools for identifying and characterizing the binary matrix representations of permutations.

Key words. shuffle-exchange network, rearrangeable network, permutations, balanced matrices, frames

AMS subject classifications. 68M07, 68M10, 05A05, 05C70

PII. S0097539798344847

1. Introduction.

An interconnection network (IN) with $N=2^n$ inputs/outputs is called a rearrangeable network if it realizes each and every one of $N!$ permutations in a single pass [9]. It is known that the lower bound for the number of stages of a symmetrical multistage IN with 2×2 switching boxes (SBs) to be rearrangeable is $2n-1$ [10]. The validity of this lower bound for shuffle-exchange (SE) networks can be established by showing the existence of a permutation which cannot be performed with less than $2n-1$ stages [2]. The question of whether or not a $(2n-1)$-stage SE network is rearrangeable has remained open for three decades [3, 4, 7, 8, 12, 13, 16, 18, 19, 21, 25, 26, 27]. This paper proves that a $(2n-1)$-stage SE network with 2×2 SBs is rearrangeable.

SE networks, initially proposed by Stone [7], provide an efficient interconnection scheme for implementing various types of parallel processes [7, 1, 6, 11, 14, 20]. These networks are constructed of repeated copies of an SE stage which consists of a "perfect shuffle" interconnection pattern followed by a column of 2×2 SBs [7, 8]. The most used SE network is the Omega network consisting of n SE stages. For about three decades, several researchers have been interested in the number of SE stages needed to realize all $N!$ permutations. The algorithm proposed by Stone [7] can be used to realize any permutation on an SE network with n^2 stages. Siegel [12] also described an algorithm for realizing any permutation on a single SE stage in $2n^2$ passes. Parker [8] showed that three passes through the Omega network are sufficient to generate any permutation, and two passes are necessary. Wu and Feng [18] have shown that a $(3n-1)$-stage SE network implements all $N!$ permutations. The upper bound of $(3n-1)$ stages was later reduced to $(3n-3)$ stages by different researchers (Huang and Tripathi [19] and Babu and Raghavendra [16]). Using a constructive approach, Raghavendra and Varma [3] showed that an SE network of five stages with $N=8$ inputs/outputs is

*Received by the editors September 16, 1998; accepted for publication (in revised form) August 12, 2002; published electronically March 5, 2003.
http://www.siam.org/journals/sicomp/32-3/34484.html
†Computer Science and Engineering Department, Arizona State University, Tempe, AZ 85287 (hasan.cam@asu.edu).

rearrangeable. This has also been proven by Linial and Tarsi [4], who provided the best previously known upper bound of $3n-4$ stages. The rearrangeability of a $(2n-1)$-stage SE network was first conjectured by Beneš, and the conjecture was published in 1975 [26, 27]. Raghavendra [21] surveyed the latest status of the rearrangeability conjecture of SE networks, along with some claims of unsuccessful proofs made earlier. Raghavendra [21] also stated that they have shown the rearrangeability of a seven-stage SE network with 16 inputs by running a program that exhaustively checks all 16! permutations. Kim, Yoon, and Maeng [25] have recently refuted the claim made by Feng and Seo [24] for proving the rearrangeability conjecture.

To prove the rearrangeability of a $(2n-1)$-stage SE network, this paper first identifies the permutations realized by the reverse baseline (RB) network, called $RB_{1:n}$, using "frames," which look like grids with horizontal links of different lengths [22]. Then it is proven that a composite network, called $RB_{1:(n-1)}SE_{1:n}$ (consisting of an $(n-1)$-stage RB network followed by an n-stage SE network), is rearrangeable. Finally, the functional equivalence of $RB_{1:(n-1)}SE_{1:n}$ and the $(2n-1)$-stage SE network is shown.

The remainder of this paper is organized as follows. The basic terminology and definitions used throughout the paper are introduced in section 2. Basic concepts and some preliminary results concerning the relationship between frames, balanced matrices, and some INs are presented in section 3. The objective of section 4 is to show the rearrangeability of $RB_{1:(n-1)}SE_{1:n}$. The proof that SE networks with at least $2n-1$ stages are rearrangeable is presented in section 5. Section 6 is dedicated to conclusions. The appendix contains some proofs that are omitted earlier.

2. Basic definitions. Throughout this paper, matrices are denoted by single capital letters, and columns of a matrix are represented by the lower case of the letter denoting that matrix. Matrix A having N rows and k columns is denoted by $A_{N\times k}$. However, the columns of a matrix are numbered starting with 1. The rows of any matrix are labeled in ascending order starting with 0, unless otherwise specified. The N rows of $A_{N\times k}$ can form $N/2$ pairs such that the row contents of each pair can be swapped/unswapped independent of the other pairs, without swapping the labels (i.e., the order of row labels in $A_{N\times k}$ is never changed). To refer to a set of specific columns of a matrix, the notation $A_{x:y}$ is used to denote the submatrix that contains those columns of A whose indices are $x,\ x+1,\dots,y$, where $1\le x\le y$; if x happens to be greater than y, then $A_{x:y}$ refers to a nil matrix. Unless specifically stated, the number of the rows of a matrix $A_{x:y}$ is assumed to be equal to N and N=2^n. $A_{N\times k}(i)$ refers to row i of the matrix $A_{N\times k}$, where $0\le i\le N-1$. A *permutation* on a set X is a bijection of X onto itself. A permutation f permutes the ordered list $0,\ 1,\dots,\ N-1$ into $f(0),\ f(1),\dots,f(N-1)$.

Definition 2.1 (permutation matrix, identity permutation matrix, reverse permutation matrix). *A permutation h is represented by an $N\times n$ binary matrix called a permutation matrix, H, such that its ith row, $H_{N\times n}(i)$, is the binary representation of the integer $h(i)$. The identity permutation matrix, denoted by $I_{N\times n}$, is the matrix whose ith row is the binary representation of i (this is called a "standard matrix" in* [15]*). The reverse permutation matrix, denoted $R_{N\times n}$, is the matrix whose jth column is the $(n+1-j)$th column of $I_{N\times n}$.*

As an example, the identity permutation matrix $I_{8\times 3}$, the reverse permutation matrix $R_{8\times 3}$, and a permutation matrix $H_{8\times 3}$ are shown below. There is a one-to-one correspondence between permutations and permutation matrices. For instance, $R_{8\times 3}$ represents the permutation $r = (0)(1\ 4)(2)(3\ 6)(5)(7)$ in cyclic notation, where we

transpose 1 and 4, transpose 3 and 6, and keep everything else fixed.

$$I_{8\times 3} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\!\!\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad R_{8\times 3} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\!\!\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad H_{8\times 3} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\!\!\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

If the binary representation of an integer i has the form $i_1 i_2 \dots i_n$, then the *shuffle* operation cyclically shifts its bits one place left, that is, $shuffle(i_1 i_2 \dots i_n) = i_2 i_3 \dots i_n i_1$. Given that the input and output addresses of an interconnection pattern with $N = 2^n$ inputs/outputs ports have the form $i_1 i_2 \dots i_n$, the *perfect shuffle* connection performs the transformation $shuffle(i_1 i_2 \dots i_n) = i_2 i_3 \dots i_n i_1$, whereas the *inverse perfect shuffle* connection performs the transformation $inverse - shuffle(i_1 i_2 \dots i_n) = i_n i_1 i_2 \dots i_{n-1}$. Let γ and γ_0 denote a general permutation and the identity permutation, respectively. Also, for $1 \le k \le n$, let γ_k denote the permutation obtained by applying the *shuffle* operation to γ_{k-1}; that is, $\gamma_k = shuffle\ (\gamma_{k-1})$. Note that γ_n also becomes the identity permutation, and γ_{n-1} is the inverse-shuffle permutation.

A k-stage IN consists of k columns of SBs, each followed and preceded by links which form interconnection patterns. A column of IN contains $N/2$ SBs with two inputs/outputs, each of which can be set both straight and cross. Figure 2.1 shows two networks considered in this paper, namely, the RB and the four-stage SE. Network stages are numbered in ascending order starting with 1 for the leftmost stage. The inputs of the leftmost stage (i.e., first stage) of any IN are labeled in ascending order from 0 to $N-1$. The inputs of the other stages of any IN except the inverse shuffle-exchange (ISE) network are also labeled from 0 to $N-1$. The inputs of ISE_k (i.e., stage k of the ISE) are labeled by the permutation γ_{k-1} described above for all i, $0 \le i \le N-1$. Specifically, the input i of ISE_k is labeled by $\gamma_{k-1}(i)$. These input labels can also be obtained as follows: (1) the inputs of the first stage ISE_1 are numbered from 0 to $N-1$ in ascending order, (2) all SBs are set straight, and (3) the input labels of the first stage are propagated through the ISE network to determine the input labels of SBs in the other stages. For instance, for $N = 16$, the inputs of ISE_2 are numbered by 0,2,4,6,8,10,12,14,1,3,5,9,11,13, and 15 from top to bottom; similarly, the inputs of ISE_3 are numbered by 0,4,8,12,1,5,9,13,2,6,10,14,3,7,11, and 15.

DEFINITION 2.2 (RB network). *An n-stage reverse baseline (RB) network is the same as the network formed by the last n stages of the Beneš network with N inputs/outputs. Specifically, for $1 \le i \le n$, stage i of the RB network consists of a pile of 2^{n-i} copies of the SE stage with 2^i inputs/outputs. (An SE stage consists of a shuffle interconnection pattern followed by a column of 2×2 SBs.) Figure* 2.1(a) *illustrates an RB network with* 16 *inputs/outputs.*

DEFINITION 2.3 (SE and ISE networks). *A shuffle-exchange (SE) network with N inputs/outputs is constructed of repeated copies of a "perfect shuffle" connection followed by a column of 2×2 SBs, where the shuffle connection implements the perfect shuffle permutation. An inverse shuffle-exchange (ISE) network is constructed of repeated copies of an ISE stage consisting of a column of 2×2 SBs followed by an*

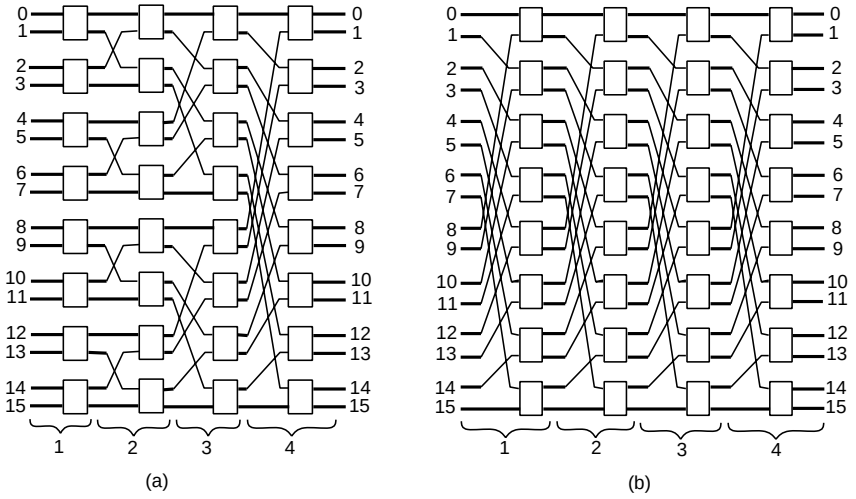


FIG. 2.1. (a) *An RB network with* 16 *inputs/outputs.* (b) *An SE network with* 16 *inputs/outputs.*

"inverse perfect shuffle" connection implementing the inverse shuffle permutation. Informally, we would obtain the ISE network if data flowed from right to left in the SE network. Figure 2.1(b) *illustrates an SE network.*

DEFINITION 2.4 (stages of RB and SE networks). *With one exception, a stage in the RB and SE networks consists of a connection pattern and the following column of SBs. The exception is the rightmost stage (i.e., the output stage), which consists of the last column of SBs and both the preceding and succeeding connection patterns. Stages are labeled from left to right in ascending order starting with* 1. *The interconnection pattern of an SE stage is called a shuffle (or perfect shuffle) interconnection pattern that implements the shuffle permutation. The interconnection pattern of stage* i *in the RB network consists of a pile of* 2^{n-i} *copies of shuffle interconnection patterns with* 2^i *inputs/outputs. See Figure* 2.1 *for the stages.*

An IN with N *inputs/outputs and* k *stages is denoted by both* $IN_{N\times k}$ *and* $IN_{1:k}$ *for* $k \geq 1$. *The subnetwork that consists of the stages* x *through* y *of* $IN_{1:k}$ *is denoted by* $IN_{x:y}$, *where* $1 \leq x \leq y \leq k$. *The notation used for networks is different from that used for matrices because matrices are always denoted by single letters, as opposed to the double letters used for INs. In this paper, if the name of an IN has more than one word, then it is denoted by the upper case form of the first letters of those words.*

DEFINITION 2.5 (RB, SE, ISE, composite IN). *The symbols* RB, SE, *and* ISE *in this paper refer to the network's reverse baseline, shuffle-exchange, and inverse shuffle-exchange, respectively. If an IN is a cascade of different INs, then it is called a composite IN and is denoted by the concatenation of symbols that represent the* IN*s in the order they are cascaded.*

As an example for a composite network, the notation $RB_{1:n}SE_{1:m}$, $m \geq 1$, denotes the network consisting of $RB_{1:n}$ followed by $SE_{1:m}$. The concept of balanced matrices was introduced by Linial and Tarsi [4]. Next, we express their definition using permutations.

DEFINITION 2.6 (balanced matrix). *Let* $N = 2^n$, $n \geq 1$, $k \geq 1$, *and* $1 \leq m \leq n$. *A matrix* $A_{N\times k}$ *containing* 0*'s and* 1*'s is balanced only if, for every set of* m

consecutive columns, every m-bit binary number appears 2^{n-m} *times in the rows. This implies that, for* $k \geq n$, *each and every n consecutive column forms the binary representation of a permutation on the set* $\{0, 1, \ldots, N-1\}$.

As an example, two balanced matrices D and E are shown below. However, notice that the matrix $[D\ E]$ is not balanced because not all three consecutive columns of $[D\ E]$ form the binary representation of a permutation on $\{0, 1, \ldots, 7\}$. For instance, "001" is repeated on rows 1 and 2 of columns 2, 3, and 4 in $[D\ E]$.

$$D = [d_1\ d_2\ d_3] = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \qquad E = [e_1\ e_2] = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

A subset M of edges in a graph G is called independent or a *matching* if no two edges of M have a vertex in common. A matching M is said to be a *perfect matching* if it covers all vertices of G. For more extended discussion of these basic concepts, the reader is referred to [5].

DEFINITION 2.7 (perfect matching graph of a balanced matrix of order $N \times (n-1)$). *Let A be a balanced matrix of order* $2^n \times (n-1)$ *for* $n \geq 2$. *The* perfect matching graph of A, *denoted by* PG_A, *on* 2^n *vertices is a graph whose vertices have degree one and vertices* V_i *and* V_j *are joined by an edge if the* i*th and* j*th rows of A are identical.*

Note that the perfect matching graph of $A_{N\times(n-1)}$ is unique because each $(n-1)$-bit row is repeated twice in $A_{N\times(n-1)}$. As an example, consider the balanced matrix E given in the above example. Its perfect matching graph is unique and has four edges such that each of the following four sets contains the two vertices of an edge: $\{V_0,\ V_1\}$, $\{V_2,\ V_3\}$, $\{V_4,\ V_5\}$, and $\{V_6,\ V_7\}$. The perfect matching graph corresponding to the first two columns of the above balanced matrix D also has four edges such that each of the following four sets contains the two vertices of an edge: $\{V_0,\ V_4\}$, $\{V_1,\ V_6\}$, $\{V_2,\ V_5\}$, and $\{V_3,\ V_7\}$.

DEFINITION 2.8 (2-labeling). *The* 2-labeling *(or* 2*-coloring) of a graph refers to the assignment of integers* 0 *and* 1 *to its vertices such that no two adjacent vertices are assigned the same integer. The integers* 0 *and* 1 *that are assigned to two adjacent vertices are called* 2*-labels.*

DEFINITION 2.9 (agree). *A balanced matrix* $B_{N\times j}$, $j \geq 1$, *is said to agree with a balanced matrix* $A_{N\times i}$, $i \geq 1$, *if the matrix* $[A_{N\times i}\ B_{N\times j}]$ *is balanced.*

LEMMA 2.10. *Given a balanced matrix* $A_{N\times(n-1)}$, $n \geq 2$, *there exist* $2^{N/2}$ *binary column vectors, say, x's, each having* $N/2$ 0*'s and* $N/2$ 1*'s such that* $[A_{N\times(n-1)}\ x]$ *is balanced for each x.*

Proof. The unique perfect matching graph, PG_A, of $A_{N\times(n-1)}$ has $N/2$ edges. There are $2^{N/2}$ distinct possible 2-labelings of this graph because each pair of adjacent vertices can be labeled in two different ways (0-1 or 1-0). Each 2-labeling corresponds to a distinct column of length N with $N/2$ 0's and $N/2$ 1's. Thus the total number of distinct columns that agree with $A_{N\times(n-1)}$ is $2^{N/2}$. □

DEFINITION 2.11 (forward-routing, reverse-routing). *Assume that a setting of SBs of an interconnection network* $IN_{N\times k}$ *realizes the permutation* $h: i \Rightarrow h(i)$ *for* $0 \leq i \leq N-1$. *When* $A(i)$ *(i.e., row i of A) is located at input i of* $IN_{N\times k}$ *for every*

i, forward-routing *matrix A through $IN_{N\times k}$ with the above setting of SBs generates a new matrix, denoted A^F, at the outputs of $IN_{N\times k}$ by permuting the rows of A in accordance with the permutation h such that $A^F(h(i)) = A(i)$. Likewise, when $A(i)$ is located at output i of $IN_{N\times k}$ for every i,* reverse-routing *matrix A through $IN_{N\times k}$ with the above setting of SBs generates a new matrix, denoted A^R, at the inputs of $IN_{N\times k}$ by permuting the rows of A in accordance with the inverse of the permutation h, denoted h^{-1}, such that $A^R(h^{-1}(i)) = A(i)$.*

In this paper, the following *bit-controlled routing scheme* is used for routing inputs of a network to their destinations. The content of row i of a balanced matrix $A_{N\times k}$ is used as the *routing tag* for the input i. Similar to the labeling scheme of network stages shown in Figure 2.1, the bits of any routing tag are also labeled from left to right, starting with 1. The ith bit of the routing tag is used as the *control bit* in the following way at the ith stage for setting an SB to which the routing tag is an input: if the control bit equals 0, then the routing tag is sent to the upper output of the SB; otherwise, it is sent to the lower output. Because an SB is allowed to be set straight and cross only in the paper, the control bits of an SB must constitute the set $\{0, 1\}$ to avoid having any conflict.

DEFINITION 2.12 (realize, pass). *A network $IN_{N\times k}$ realizes a permutation represented by $A_{N\times n}$ if there is a network switch setting such that input i is sent to output $A_{N\times n}(i)$ for all $i = 0, 1, \ldots, N-1$. A network $IN_{N\times k}$, $k \geq 1$, passes a balanced matrix $B_{N\times k}$ if no conflict occurs in the SBs of the $IN_{N\times k}$ when $B_{N\times k}(i)$ is used as the routing tag in the way explained above for the ith input of the $IN_{N\times k}$.*

3. Frames and preliminary results. This section introduces the notion of a frame and establishes necessary fundamental properties of frames and how they relate to balanced matrices and INs of interest. Somewhat different versions of some of these definitions and a motivational example appeared in [22, 23]. Some preliminary results that will be used throughout the paper are also presented in this section.

Recall that γ and γ_0 denote a general permutation and the identity permutation, respectively. Also, for $1 \leq k \leq n$, let γ_k denote the permutation obtained by applying the *shuffle* operation to γ_{k-1}, that is, $\gamma_k = shuffle(\gamma_{k-1})$.

A frame looks like a grid with horizontal links of different lengths together with a labeling of the rows and columns. The frame is used to capture constraints that some rows and columns of balanced matrices need to satisfy in order to be realized by INs.

DEFINITION 3.1 (frame, standard frame, block of a frame). *A frame $F_{N\times k;\gamma}$ consists of k columns labeled f_j for $1 \leq j \leq k$ and $k \geq 1$ from left to right and N rows labeled according to a permutation γ such that the label of the ith row from the top equals $\gamma(i)$ for all i, $0 \leq i \leq N-1$. For $1 \leq j \leq n-1$, column f_j consists of 2^{n-j} blocks of 2^j rows (entries) each. For $j \geq n$, column f_j consists of a single block of N rows. If γ is the identity permutation (i.e., γ_0), then the frame is called the standard frame and may also be denoted by $F_{N\times k}$, in addition to the notation $F_{N\times k;\gamma_0}$. In the graphical representation of a frame, any polygon with four sides and four right angles is called a block of a frame. (See Figure* 3.1 *for some frames.)*

The notation f_{ij} denotes the ith entry of column f_j. The notation $F_{N\times k;\gamma}(i)$ is used to denote the ith row of $F_{N\times k;\gamma}$. If the number of rows in a frame is not specified, it is assumed to be equal to N. Therefore, if the number of rows is not specified, the notation $F_{N\times k;\gamma}$ and $F_{1:k;\gamma}$ refer to the same frame.

DEFINITION 3.2 ($F^r_{N\times k;\gamma}$, universal frame $F^{n-1}_{N\times k;\gamma}$). *Frame $F^r_{N\times k;\gamma}$, $r \in \{0, 1, \ldots, k-1\}$ and $k \in \{1, 2, \ldots, n\}$, is the same as $F_{N\times k;\gamma}$ except that for $j = 1, 2, \ldots, r+1$, each column f_j of $F^r_{N\times k;\gamma}$ consists of 2^{n-r-1} blocks of 2^{r+1} rows each (instead of*

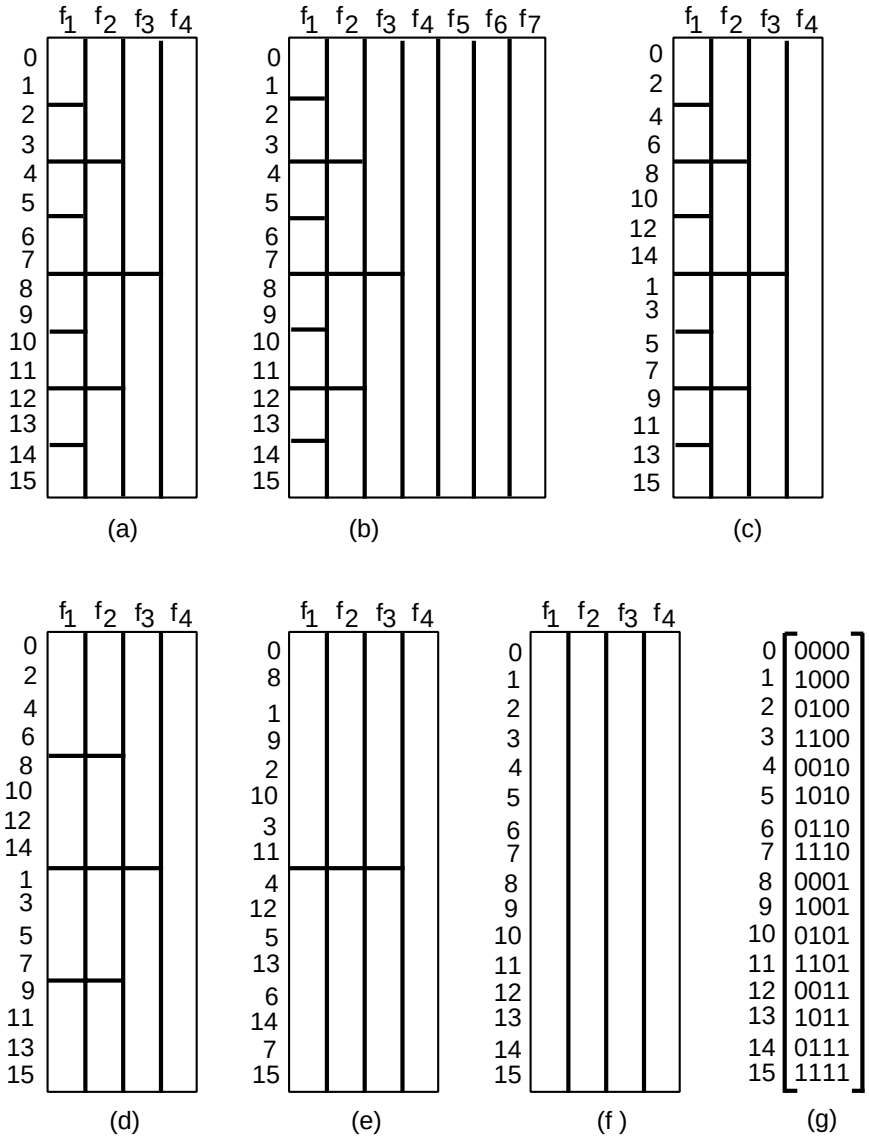


FIG. 3.1. *Let* $N = 16$. (a) $F_{16\times 4} = F^0_{1:4}$. (b) $F_{16\times 7} = F^0_{1:7}$. (c) $F_{16\times 4;\gamma_1} = F^0_{1:4;\gamma_1}$. (d) $F^1_{16\times 4;\gamma_1} = F^1_{1:4;\gamma_1}$. (e) $F^2_{16\times 4;\gamma_3} = F^2_{1:4;\gamma_3}$. (f) *The universal frame* $F^3_{16\times 4;\gamma_0} = F^3_{1:4}$. (g) *The reverse permutation matrix* $R_{16\times 4} = R_{1:4}$.

2^{n-j} *blocks of* 2^j *rows each). For* $r = n-1$, *the frame* $F^{n-1}_{N\times k;\gamma}$ *is called the universal frame.*

Note that every column of the universal frame is a single block of N rows. (See Figure 3.1 for some $F^r_{N\times k;\gamma}$ and the universal frame.) Also, note that $F_{N\times k;\gamma}$ and $F^0_{N\times k;\gamma}$ refer to the same frame.

DEFINITION 3.3 (fit). *Let* $k \geq 1$ *and* $1 \leq p \leq k$. *Consider a matrix* $A_{N\times k}$ *and a frame* F *with* N *rows and* k *columns. First, row* i *of* $A_{N\times k}$ *is placed into the row*

i of F. Then the matrix $A_{N\times k}$ is said to fit the frame F if and only if, for all p's, each and every one of those p-column blocks whose entries are located in columns f_1, $f_2, \ldots, f_p$ of F contains a balanced matrix.

Note that any balanced matrix of order $N \times k$ fits a universal frame with N rows and k columns because a universal frame requires a matrix to be balanced only.

Example 3.1. Note that the reverse permutation matrix $R_{16\times 4}$ shown in Figure 3.1(g) fits $F_{16\times 4}$ shown in Figure 3.1(a). However, $R_{16\times 4}$ does not fit $F_{16\times 4;\gamma_1}$ shown in Figure 3.1(c) because many blocks of $F_{16\times 4;\gamma_1}$ do not contain balanced matrices when row i of $F_{16\times 4;\gamma_1}$ is filled in by row i of $R_{16\times 4}$. For instance, in $F_{16\times 4;\gamma_1}$, the topmost block of f_1 does not contain a balanced matrix.

In the remainder of this section, some preliminary results are presented. The following lemma computes the number of matrices that fit $F_{N\times k}$.

LEMMA 3.4. *The number of those matrices of order $N \times k$ each of which fits $F_{N\times k}$ equals $2^{Nk/2}$ for $k \geq 1$.*

Proof. The first column f_1 of $F_{N\times k}$ consists of $N/2$ blocks each having only two rows. Because each block can constitute the set $\{0, 1\}$ in two different ways independently of the other blocks, there exist $2^{N/2}$ column vectors that fit the first column of $F_{N\times k}$. Each of these column vectors, say, x, generates $2^{N/2}$ column vectors, say, y's, in the sense that each of $[x\ y]$ fits the first two columns of $F_{N\times k}$. To select a vector y for column x, $N/2$ pairs of rows are first determined for x such that two rows of x form a pair if their contents are the same and their second entries on column y will belong to the same block of f_2 of $F_{N\times k}$. The rows of each pair are swapped/unswapped independently of the other pairs of rows. Thus each x generates $2^{N/2}$ y's such that any $[x\ y]$ fits the first two columns of $F_{N\times k}$. Similarly, to select a vector z for a given balanced matrix $[x\ y]$, $N/2$ pairs of rows are first determined for the matrix $[x\ y]$ such that two rows of $[x\ y]$ form a pair if their contents are the same and their third entries on column z will belong to the same block of f_3. The rows of each pair are swapped/unswapped independently of the other pairs of rows. Thus each $[x\ y]$ generates $2^{N/2}$ z's such that any $[x\ y\ z]$ fits the first three columns of $F_{N\times k}$. This generation of column vectors is continued until all matrices that fit $F_{N\times k}$ are generated. Thus the lemma holds. ∎

The following theorem establishes a relation between frames and permutations that pass RB networks. The basic idea behind the proof of the theorem appears in [22], and the complete proof is provided in the appendix.

THEOREM 3.5. *A matrix $D_{1:k} = [d_1\ d_2\ \ldots\ d_k]$ fits $F_{1:k}$ if and only if $D_{1:k}$ passes $RB_{1:k}$, $1 \leq k \leq n$. Moreover, $RB_{1:k}$ sends its ith input to its jth output, where j is equal to the sum of $(\lfloor i/2^k \rfloor \times 2^k)$ and the value of $D_{1:k}(i)$.*

Note that the sum described in Theorem 3.5 is a regular sum in the sense that no modular arithmetic is needed. Theorem 3.5 implies that, for $k = n$, a matrix $D_{1:k}$ fits $F_{1:k}$ if and only if $D_{1:k}$ is *realized* by $RB_{1:k}$. It is shown in [4] how balanced matrices can be used to determine the number of SE stages (or the number of passes through a single SE stage) necessary to realize a given permutation. Lemma 3.6 below restates their result using the notation of this paper.

LEMMA 3.6 (see [4]). *Let $M_{N\times m}$ and $C_{N\times k}$ be balanced matrices such that $M_{N\times m} = [I_{N\times n}\ C_{N\times k}]$, $k \geq 1$, and $n + k = m$. The k-stage SE network (i.e., $SE_{1:k}$) realizes the permutation represented only by $M_{(m+1-n):m}$.*

To illustrate Lemma 3.6, consider the identity permutation matrix $I_{8\times 3} = [i_1\ i_2\ i_3]$ and some balanced matrices $M_{8\times 4} = [I_{8\times 3}\ i_1]$, $M_{8\times 5} = [I_{8\times 3}\ i_1\ i_2]$, and $M_{8\times 6} = [I_{8\times 3}\ I_{8\times 3}]$. Because $M_{8\times 4}$, $M_{8\times 5}$, and $M_{8\times 6}$ are balanced, the permutations repre-

sented in binary by $[i_2\ i_3\ i_1]$, $[i_3\ i_1\ i_2]$, and $[i_1\ i_2\ i_3]$ are realized by the single-stage SE, two-stage SE, and three-stage SE with $N = 8$ inputs/outputs, respectively.

Lemma 3.6 can be used to establish the *lower bound* of $2n - 1$ stages for SE networks as follows. Since $M_{1:m}$ defined in Lemma 3.6 is a balanced matrix, every n consecutive columns of it form a balanced matrix. Even if only one column vector of a balanced matrix is already known, no other column of the balanced matrix can be an arbitrary column vector because each one of them has to form a balanced submatrix with the fixed column. Therefore, in order for $M_{(m+1-n):m}$ to represent any balanced matrix of order $N \times n$, it is required that the rightmost column i_n of $I_{1:n}$ does not form a balanced submatrix with any column of $M_{(m+1-n):m}$. This implies that there must be at least $n-1$ column vectors between i_n and the leftmost column $m+1-n$ of $M_{(m+1-n):m}$, which implies that $m+1-n$ must be equal to at least the sum of n and the number of columns of $I_{1:n}$, that is, $m+1-n \geq 2n$. Hence $m \geq 3n-1$. Since the k-stage SE network realizes the permutation represented by $M_{(m+1-n):m}$, where $k = m - n$, k must be at least $2n - 1$. Therefore, an SE network must have at least $2n - 1$ stages to be rearrangeable.

The following theorem shows that $F_{1:(n+m-1)}$ can be used to characterize the permutations realized by $RB_{1:(n-1)}SE_{1:m}$ for $m \geq 1$. The proof of this theorem is provided in the appendix.

THEOREM 3.7. *A balanced matrix $D_{1:(n+m-1)}$, $m \geq 1$, fits the frame $F_{1:(n+m-1)}$ if and only if $D_{1:(n+m-1)}$ passes the network $RB_{1:(n-1)}SE_{1:m}$. Moreover, $RB_{1:(n-1)}SE_{1:m}$ realizes the permutation represented by the last n columns of $D_{1:(n+m-1)}$, denoted by $D_{m:(n+m-1)}$.*

Now we introduce Figure 3.2 and explain below the basic idea behind Theorem 3.7. Let $(x_1^i x_2^i \dots x_n^i \dots x_{n+m-1}^i)$ denote the row i of a balanced matrix $D_{1:(n+m-1)}$ fitting $F_{1:(n+m-1)}$, where $0 \leq i \leq N-1$. As stated above, Theorem 3.5 implies that, for $k = n$, a matrix $D_{1:k}$ fits $F_{1:k}$ if and only if $D_{1:k}$ is *realized* by $RB_{1:k}$. Thus any permutation whose binary representation fits $F_{1:n}$ is realized by $RB_{1:n}$, and vice versa. This implies that when $(x_1^i x_2^i \dots x_n^i)$ is used as the routing tag of input i of $RB_{1:n}$, this routing tag (along with input i) reaches the output with label $(x_1^i x_2^i \dots x_n^i)$ of $RB_{1:n}$. (Note that the bit x_j^i, $1 \leq j \leq n$, is used as the control bit for setting SB at stage j of $RB_{1:n}$.) This also means that the matrix formed by all the N routing tags $(x_1^i x_2^i \dots x_n^i)$ at the outputs of $RB_{1:n}$ is the identity permutation matrix. Recall that the last stage of $RB_{1:n}$ is an SE stage, and hence $RB_{1:n}$ is identical to $RB_{1:(n-1)}SE_1$. Therefore, it follows that the permutation whose binary representation is the same as the submatrix $D_{1:n}$ of $D_{1:(n+m-1)}$ is realized by $RB_{1:(n-1)}SE_1$, and all the N routing tags $(x_1^i x_2^i \dots x_n^i)$ form the identity permutation matrix at the outputs of $RB_{1:(n-1)}SE_1$.

When an SE stage is appended to the right of $RB_{1:(n-1)}SE_1$ in order to form $RB_{1:(n-1)}SE_{1:2}$, the routing tag $(x_1^i x_2^i \dots x_n^i)$ is first shuffled through the shuffle pattern to the output with label $(x_2^i x_3^i \dots x_n^i x_1^i)$ of the shuffle pattern and is then switched by the exchange stage to the output with label $(x_2^i x_3^i \dots x_n^i x_{n+1}^i)$ of $RB_{1:(n-1)}SE_{1:2}$, where x_{n+1}^i is used as the control bit for setting the SB of the exchange stage. Because Theorem 3.7 guarantees that no conflict occurs in the SBs, the matrix formed by all the N routing tags $(x_2^i x_3^i \dots x_n^i x_{n+1}^i)$ at the outputs of $RB_{1:(n-1)}SE_{1:2}$ is now the identity permutation matrix. This implies that the permutation whose binary representation is the same as the submatrix $D_{2:(n+1)}$ is realized by $RB_{1:(n-1)}SE_{1:2}$. In general, when an SE stage is appended to the right of $RB_{1:(n-1)}SE_{1:(m-1)}$ in order to form $RB_{1:(n-1)}SE_{1:m}$, the routing tag $(x_{m-1}^i x_m^i \dots x_{n+m-2}^i)$ is first shuffled

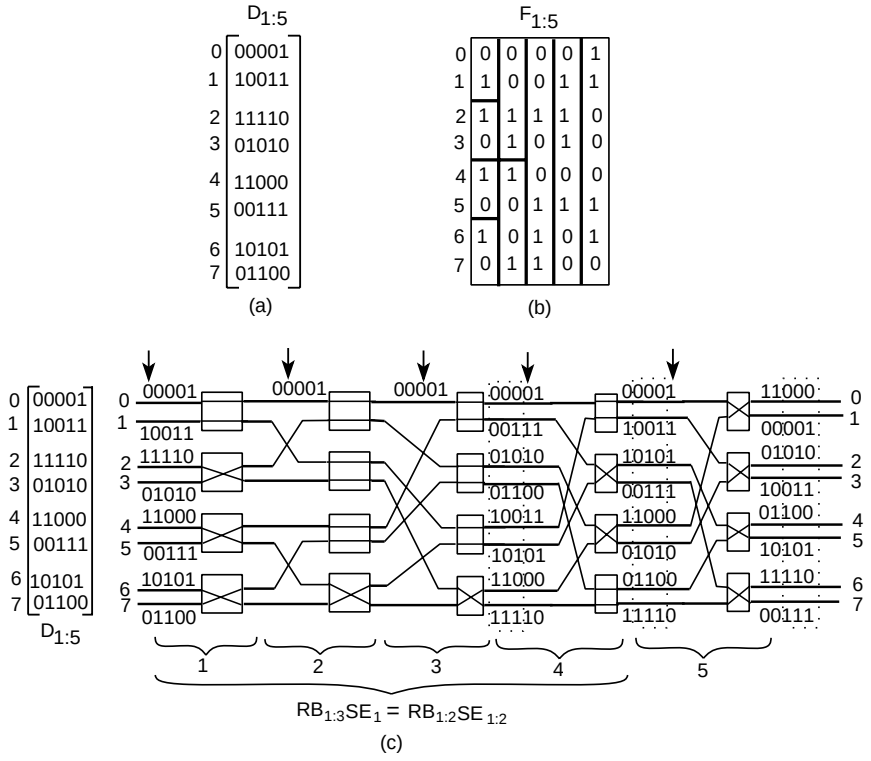


FIG. 3.2. *In this figure,* $N = 8$, $n = 3$, *and* $m = 3$. (a) *A balanced matrix* $D_{1:5}$ *fitting* $F_{1:5}$. (b) $F_{1:5}$ *along with the matrix* $D_{1:5}$. (c) *The network* $RB_{1:2}SE_{1:3}$ *realizes the permutation corresponding to the balanced submatrix* $D_{3:5}$ *(i.e., the last* n *columns of* $D_{1:5}$*), where the row* i *of* $D_{1:5}$ *is used as the routing tag of the input* i. *Arrows point to the control bits used to set the SBs. The matrices formed by the routing tags at the stages* 4, 5 *and the outputs of* $RB_{1:2}SE_{1:3}$ *are shown, and each identity permutation matrix is encircled by a dotted line box. The row labels of matrices that are formed in every stage of the network are in ascending order from* 0 *to* 7.

through the shuffle pattern to the output with label $(x_m^i x_{m+1}^i \dots x_{n+m-2}^i x_{m-1}^i)$ of the shuffle pattern and is then switched by the exchange stage to the output with label $(x_m^i x_{m+1}^i \dots x_{n+m-2}^i x_{n+m-1}^i)$ of $RB_{1:(n-1)}SE_{1:m}$. Because the matrix formed by all the N routing tags $(x_m^i x_{m+1}^i \dots x_{n+m-2}^i x_{n+m-1}^i)$ at the outputs of $RB_{1:(n-1)}SE_{1:m}$ is the identity permutation matrix, the permutation whose binary representation is the same as the submatrix $D_{m:(n+m-1)}$ is realized by $RB_{1:(n-1)}SE_{1:m}$. In other words, the permutation whose binary representation is the same as the last n columns of $D_{1:(n+m-1)}$ is realized by $RB_{1:(n-1)}SE_{1:m}$.

For $N = 8$ and $m = 3$, Figure 3.2 shows how the rows of a $D_{1:(n+m-1)}$ are used as the routing tags of the inputs of $RB_{1:(n-1)}SE_{1:m}$.

This figure also illustrates how the permutation corresponding to the last n columns of $D_{1:(n+m-1)}$ is realized by $RB_{1:(n-1)}SE_{1:m}$. The matrices formed by the routing tags at the third, fourth, and fifth stages of $RB_{1:2}SE_{1:3}$ are shown such that each identity permutation matrix formed by the routing tags is encircled by a dotted line box. The dotted line box at the outputs of $RB_{1:2}SE_{1:3}$ contains the last three columns of the matrix formed by the routing tags. This implies that the permutation

whose binary representation is $D_{3:5}$ is realized by $RB_{1:2}SE_{1:3}$.

4. Rearrangeability of network $RB_{0:(n-1)}SE_{1:n}$. The main result of this section, stated in Theorem 4.7, is that $RB_{1:(n-1)}SE_{1:n}$ is rearrangeable. This result is obtained as follows. Let T denote the set of all those balanced matrices of order $N \times (2n-1)$, each of which fits $F_{1:(2n-1)}$. Also, let M denote the set of all those balanced matrices of order $N \times n$, each of which is the same as the last n columns of at least one member of T. Theorem 3.7 states that $RB_{1:(n-1)}SE_{1:n}$ realizes every permutation whose binary representation is the same as the last n columns of a matrix fitting $F_{1:(2n-1)}$, and vice versa. Therefore, to prove that $RB_{1:(n-1)}SE_{1:n}$ is rearrangeable, it is sufficient to show that the set M contains all $N!$ balanced matrices of order $N \times n$. To show this, we introduce three algorithms, namely, COLUMN, GENERATE-π, and GENERATE-ALL. Algorithm GENERATE-ALL generates all matrices of T in $n-1$ steps. In each step, a column vector r of $N/2$ 0's followed by $N/2$ 1's is added to the left of every balanced matrix that is generated in the previous step. (In step 1, the column vector is added to the left of every one of $N!$ balanced matrices.) Since prepending the column vector makes some resulting matrices unbalanced, those unbalanced matrices are removed, and each of the remaining balanced matrices is reverse-routed through an ISE stage of $N/2$ switches in $2^{N/2}$ ways to generate new matrices. This reverse-routing is done in the same way as described in algorithms COLUMN and GENERATE-π. Algorithm COLUMN helps prove that every balanced matrix of order $N \times n$ is contained in the last n columns of at least one matrix generated by the reverse-routing. Algorithm GENERATE-π shows that the first $n-1$ columns of every generated matrix fit $F_{1:(n-1)}$.

Now we introduce algorithm COLUMN. Given a balanced matrix $B_{N\times(n-1)}$, $n \geq 1$ and $N = 2^n$, and $F_{N\times 1}$ (i.e., the first column of $F_{N\times n}$), algorithm COLUMN shown in Figure 4.1 determines a column vector V such that V fits $F_{N\times 1}$ and the matrix $[V\ B_{N\times(n-1)}]$ is balanced. Because $F_{N\times 1}$ consists of 2^{n-1} blocks of two rows each, the $2i$th and $(2i+1)$th entries of V constitute the set $\{0,1\}$ and hence do not cause any conflict when they become the control bits of a switch for $0 \leq i \leq 2^{n-1}-1$. Therefore, it is shown in Lemma 4.1 that, for $1 \leq k \leq n-1$, a stage of $N/2$ switches can partition any given balanced matrix $B_{N\times k}$ into two balanced matrices of order $2^{n-1} \times k$, one of which consists of only those rows of $B_{N\times k}$ sent to the upper outputs of switches, and the other matrix contains only the rows sent to the lower outputs. Corollary 4.2 shows that the reverse order of this partitioning process from the outputs side of the stage to the inputs side allows any $B_{N\times k}$ to be generated. After the proofs of Lemma 4.1 and Corollary 4.2, an example is introduced to help clarify these proofs and algorithm COLUMN.

The correctness proof of algorithm COLUMN is as follows. By the definition of perfect matching, each vertex of a perfect matching graph is incident with one edge only. Therefore, each vertex of the union of two perfect matching graphs is incident with two edges that belong to different perfect matching graphs. (The union of two perfect matching graphs may be regarded as a multigraph.) This means that each perfect matching graph has an equal number of edges in every cycle of the union graph. Therefore, each cycle contains an even number of edges, and hence 2-labeling every cycle is always possible. Because the two integers assigned to the vertices of an edge in 2-labeling constitute the set $\{0,1\}$, the vector V, whose ith entry equals the 2-label of the vertex i, agrees with $B_{1:(n-1)}$ and fits $F_{N\times 1}$.

LEMMA 4.1. *Consider a frame $F^{n-2}_{N\times k;\gamma}$ which is a pile of two universal frames, namely, upper and lower universal frames denoted UF and LF, respectively, each*

Algorithm COLUMN

Input: A balanced matrix $B_{N\times(n-1)}$ and $F_{N\times 1} = f_1$.

Output: A vector V that fits $F_{N\times 1}$ and a balanced matrix $[V\ B_{N\times(n-1)}]$.

begin

Step 1. Determine the perfect matching graph of $B_{1:(n-1)}$ by joining an edge between any two vertices that correspond to two identical rows of $B_{1:(n-1)}$.

Step 2. Determine the perfect matching graph of $F_{N\times 1}$ by joining an edge between any two vertices with labels $2m$ and $2m+1$ for $m = 0, 1, \ldots, 2^{n-1}-1$.

Step 3. 2-label the union of the perfect matching graphs of $B_{1:(n-1)}$ and $F_{N\times 1}$.

Step 4. Let the integer assigned to the vertex i in 2-labeling be the ith entry of the vector V (i.e., $V(i)$) for $0 \le i \le N-1$.

end

FIG. 4.1. *Algorithm COLUMN.*

having $N/2$ rows and k columns, where $N = 2^n$ and $1 \le k \le n-1$. Let ISE_j denote an inverse shuffle-exchange (ISE) stage with N inputs/outputs and $N/2$ switches of size 2×2 each, where $j \ge 1$. Then, for any given balanced matrix $B_{N\times k}$ fitting the universal frame *$F^{n-1}_{N\times k}$, there exists a switch setting of ISE_j such that, when $B_{N\times k}$ is forward-routed through ISE_j, a matrix fitting $F^{n-2}_{N\times k;\gamma}$ is obtained at the outputs side of ISE_j.*

Proof. We shall first prove the lemma for $k = n-1$. Recall that any balanced matrix $B_{N\times(n-1)}$ fits the universal frame $F^{n-1}_{N\times(n-1);\gamma}$. For a given balanced matrix $B_{N\times(n-1)}$, algorithm COLUMN finds a vector V of N entries such that V fits $F_{N\times 1}$ and the matrix $[V\ B_{N\times(n-1)}]$ is balanced. Let the ith entry, $0 \le i \le N-1$, of V be the control bit of the ith input of ISE_j. Thus, because V fits $F_{N\times 1}$, two control bits of each switch constitute the set $\{0,1\}$. Assume that B^u and B^l denote the matrices consisting of those rows of $B_{N\times(n-1)}$ sent to the upper and lower outputs, respectively, of switches. So, if $V(i) = 0$ (respectively, $V(i) = 1$), then row i of $B_{N\times(n-1)}$ will be forward-routed to the upper (respectively, lower) output of a switch. Because $[V\ B_{N\times(n-1)}]$ is also balanced, B^u (respectively, B^l) form a balanced matrix of order $2^{n-1}\times(n-1)$ that fits UF (respectively, LF). Therefore, the matrix obtained at the outputs of ISE_j fits $F^{n-2}_{N\times(n-1);\gamma}$.

Now, we shall prove the lemma for $1 \le k \le n-2$. It is shown above that the lemma holds for $k = n-1$. In the above proof, any $B_{N\times(n-1)}$ is forward-routed through ISE_j. Now, for any matrix $B_{N\times(n-1)}$ that was forward-routed through ISE_j, remove its rightmost $n-k-1$ columns, so that $B_{N\times(n-1)}$ is converted to a matrix $B_{N\times k}$. Similarly, remove the rightmost $n-k-1$ columns of B^u, B^l, and $F^{n-2}_{N\times(n-1);\gamma}$. Notice that the switch settings of ISE_j are not changed at all while these columns are being removed. Therefore, when any balanced matrix $B_{N\times k}$ that fits a *universal frame* $F^{n-1}_{N\times k}$ is forward-routed through ISE_j, the matrix that was obtained at the outputs side of ISE_j fits $F^{n-2}_{N\times k;\gamma}$. □

COROLLARY 4.2. *Consider the stage ISE_j and $F^{n-2}_{N\times k;\gamma}$, described in Lemma* 4.1, *where $N = 2^n$ and $1 \le k \le n-1$. Let T denote the set of all those matrices that fit $F^{n-2}_{N\times k;\gamma}$. When each and every member of T is reverse-routed through ISE_j in all possible $2^{N/2}$ ways, all balanced matrices of order $N \times k$ that fit a universal frame $F^{n-1}_{N\times k;\gamma}$ are obtained at the inputs side of ISE_j. (In reverse-routing of a member, the* contents *of rows are swapped/unswapped in every switch independently of the other switches; this is also true for forward-routing.)*

Proof. The proof is by construction. Let $B_{N\times k}$ be an arbitrary matrix that fits a universal frame $F^{n-1}_{N\times k;\gamma}$. $B_{N\times k}$ can be constructed by reverse-routing an element of T through a setting of switches of ISE_j. This construction is described below using Lemma 4.1.

Lemma 4.1 has shown that there exists a switch setting of ISE_j such that, when $B_{N\times k}$ is forward-routed through ISE_j, a matrix, say, $C_{N\times k}$, fitting $F^{n-2}_{N\times k;\gamma}$ is obtained at the outputs side of ISE_j. This implies that, when $C_{N\times k}$ is reverse-routed through the same switch setting of ISE_j, $B_{N\times k}$ is obtained at the inputs side of ISE_j. Because $C_{N\times k}$ fits $F^{n-2}_{N\times k;\gamma}$, $C_{N\times k}$ must be a member of T because T denotes the set of all those matrices that fit $F^{n-2}_{N\times k;\gamma}$. Therefore, when $C_{N\times k}$ is reverse-routed through ISE_j in all possible $2^{N/2}$ ways, one of the matrices obtained at the inputs side of ISE_j is identical to $B_{N\times k}$. □

Example 4.1. Let $N = 16$. Consider $B_{1:3}$ shown in Figure 4.2(c). Recall that any balanced matrix of order 16×3 fits the universal frame $F^3_{1:3}$ shown in Figure 4.2(a), where row i of $B_{1:3}$ is placed into row i of the universal frame. Because the *control* (or routing) bits of each switch of ISE_j must constitute the set $\{0, 1\}$, the input labels of each switch are the labels of the adjacent vertices of a distinct edge in the perfect matching graph of the switches of ISE_j, denoted PG_{ISE} and shown in Figure 4.2(e). The perfect matching graph of $B_{1:3}$ is shown in Figure 4.2(d). The union of PG_B and PG_{ISE}, denoted $PG_{B\bigcup ISE}$, and its 2-labeling are shown in Figure 4.2(f). The ith entry of vector V obtained from the 2-labeling is the control bit of the ith input of ISE_j. Vector V and the switch settings of ISE_j with respect to vector V are shown in Figures 4.2(g) and 4.2(h), respectively. When row i of the matrix $B_{1:3}$ is forward-routed through ISE_j according to the setting of the switch with input i, the resulting matrix fits the frame $F^2_{1:3;\gamma}$ shown in Figure 4.2(h). Note that the upper (respectively, lower) input and the upper (respectively, lower) output of a switch have the same label. It follows from Corollary 4.2 that, when the resulting matrix fitting $F^2_{1:3;\gamma}$ is reverse-routed through ISE_j with the same switch settings, the original matrix $B_{1:3}$ is obtained at the inputs side of ISE_j.

Recall that $R_{N\times n}$ (or $R_{1:n}$) denotes the reverse permutation matrix. So $R_{1:(n-1)}$ denotes a matrix consisting of the first $n-1$ columns of $R_{1:n}$. Lemma 4.3 shows that $R_{1:(n-1)}$ fits $F_{1:(n-1)}$. Figure 4.3 illustrates $R_{1:(n-1)}$ and $F_{1:(n-1)}$ for $N = 16$.

LEMMA 4.3. *$R_{1:(n-1)}$ fits $F_{1:(n-1)}$.*

Proof. Let $1 \le j \le n-1$. Let us partition $R_{1:j}$ into 2^{n-j} submatrices of 2^j rows and j columns each, denoted $R^h_{1:j}$ for $h = 0, 1, \ldots, 2^{n-j}-1$, such that the row labels of $R^h_{1:j}$ consist of all the numbers ranging from $h\times 2^j$ to $(h+1)2^j - 1$ inclusive. Because each $R^h_{1:j}$ is a balanced matrix of order $2^j \times j$, $R_{1:(n-1)}$ fits $F^0_{1:(n-1)}$. □

4.1. Algorithm GENERATE-π. Let π denote the set of all those balanced matrices of order $N\times(n-1)$, each of which fits $F_{1:(n-1)}$. Also, let r_k denote the kth column of the reverse permutation matrix $R_{1:(n-1)} = [r_1\, r_2 \ldots r_{n-1}]$, where $0 \le k \le n-1$. We now introduce an algorithm called GENERATE-π, shown in Figure 4.4, to gener-

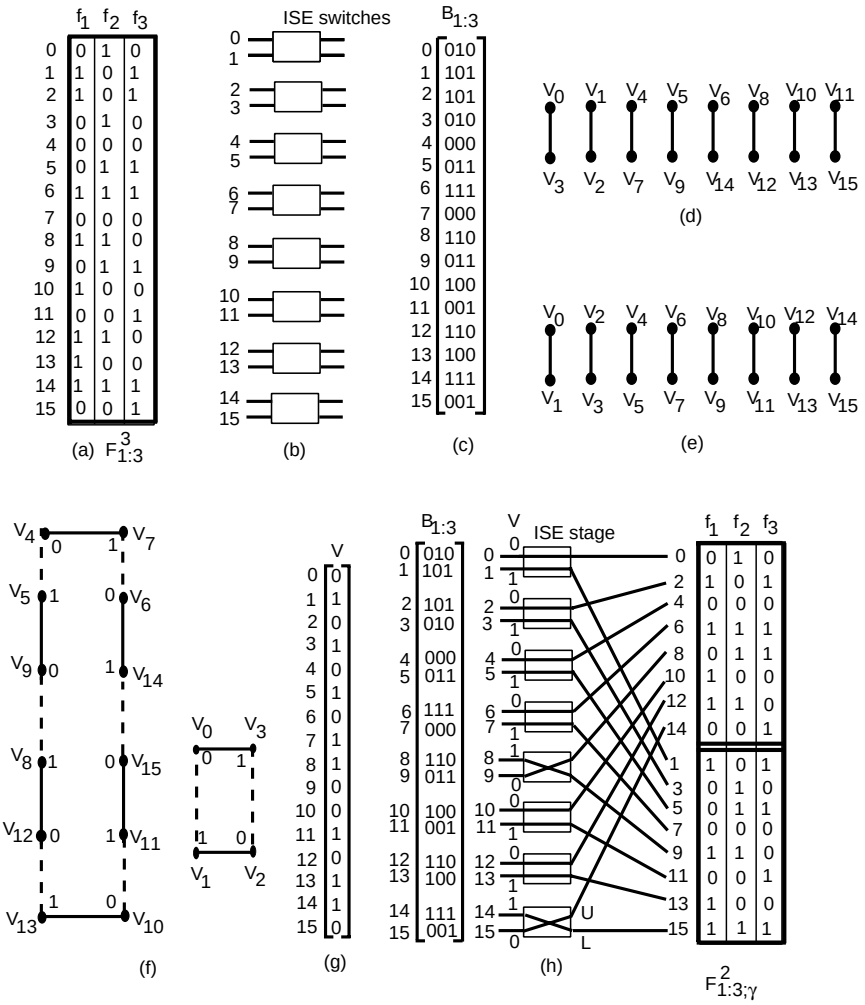


FIG. 4.2. (a) *A universal frame* $F^3_{1:3}$ *whose rows are filled in by a balanced matrix* $B_{1:3}$ *for* $N = 16$. (b) *The* $N/2$ *switches of an inverse shuffle-exchange stage* ISE_j. (c) $B_{1:3}$, *a balanced matrix.* (d) PG_B, *the perfect matching graph of* $B_{1:3}$. (e) PG_{ISE}, *the perfect matching graph of switches of* ISE_j. (f) $PG_{B \bigcup ISE}$, *where solid and dashed lines represent the edges of* PG_B *and* PG_{ISE}, *respectively; a possible* 2*-labeling of* $PG_{B \bigcup ISE}$ *is shown.* (g) *Vector* V. (h) *Switch settings of* ISE *with respect to* V. *The matrix that is obtained at the outputs side by forward-routing* $B_{1:3}$ *fits* $F^2_{1:3;\gamma}$.

ate all matrices of π in $n-1$ steps. The kth iteration of the *for* loop of GENERATE-π is referred to as the kth step. In each step, $N/2$ pairs of rows are formed. All those matrices fitting $F_{1:(n-1)}$ are generated in $n-1$ steps by swapping/unswapping the contents of each pair's rows independently of the other pairs' rows. Initially, r_{n-1} is placed into f_{n-1} of $F_{1:(n-1)}$. At the end of step k, column r_{n-k-1} of $R_{1:(n-1)}$ is inserted into column f_{n-k-1} of $F_{1:(n-1)}$, which amounts to saying that r_{n-k-1} is placed to the left of every matrix that has been generated so far. In step 1, all the generated matrices have single columns; that is, they are indeed just column vectors. In step 2, all matrices have two columns, and in step $n-1$ every matrix has $n-1$ columns.

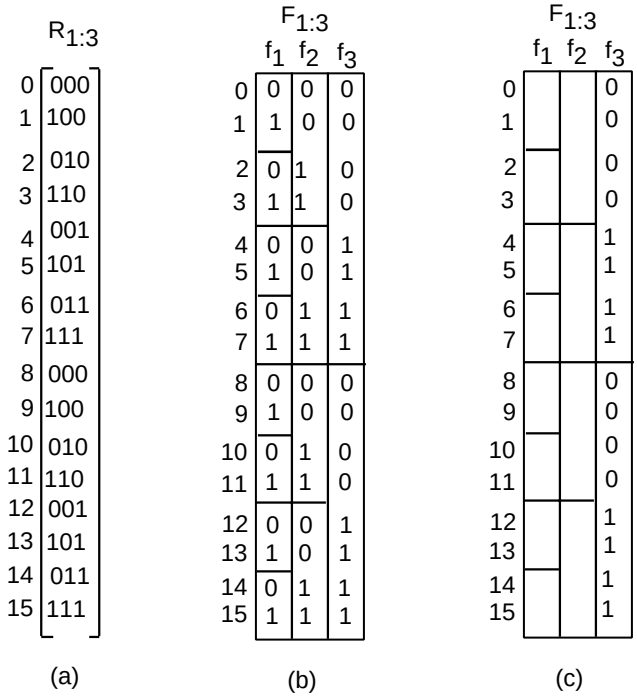


FIG. 4.3. $N = 16$. (a) $R_{1:3}$, *the first three columns of the reverse permutation matrix.* (b) $F_{1:3}$ *containing* $R_{1:3}$. (c) $F_{1:3}$ *containing the third column of* $R_{1:3}$.

As an alternative way of generating all matrices fitting $F_{1:(n-1)}$, we could have initially placed all columns of $R_{1:(n-1)}$ into $F_{1:(n-1)}$ and then generated all $2^{N(n-1)/2}$ matrices fitting $F_{1:(n-1)}$. However, the above method is adopted in algorithm GENERATE-π since algorithm GENERATE-ALL (to be introduced later in this section) uses the same method in generating the first $n-1$ columns of matrices. An example is provided first to illustrate the algorithm for $N = 16$, and then Lemma 4.4 is presented to show its correctness.

Example 4.2. This example describes how algorithm GENERATE-π works for $N = 16$. Initially, both π_0 and $F_{1:3}$ contain only r_3 of $R_{1:3}$. Figure 4.3(c) illustrates that column f_3 of $F_{1:3}$ contains r_3. The sets $\pi_1, \pi_2, \ldots, \pi_{n-1}$ are initially empty sets. In the *first step*, each of the following eight sets contains the row labels of a pair of r_3: $\{0,4\}$, $\{1,5\}$, $\{2,6\}$, $\{3,7\}$, $\{8,12\}$, $\{9,13\}$, $\{10,14\}$, and $\{11,15\}$. By swapping and unswapping the rows' *contents* of each pair independently of the contents of the other pairs of rows, r_3 generates 2^8 matrices of single columns. These generated matrices are inserted into π_1. In line 12 of GENERATE-π, column r_2 is placed to the left of each and every matrix of π_1.

In the *second step*, each of the following eight sets contains the row labels of a pair of any matrix of π_1: $\{0,2\}$, $\{1,3\}$, $\{4,6\}$, $\{5,7\}$, $\{8,10\}$, $\{9,11\}$, $\{12,14\}$, and $\{13,15\}$. By swapping and unswapping the rows' contents of each pair independently

Algorithm GENERATE-π

Input: $R_{1:(n-1)}$ and $F_{1:(n-1)}$.

Output: All balanced matrices fitting $F_{1:(n-1)}$.

begin

1. Let π_0 denote a set containing only column r_{n-1} of $R_{1:(n-1)}$.
2. Insert column r_{n-1} of $R_{1:(n-1)}$ into column f_{n-1} of $F_{1:(n-1)}$.
3. **for** $k \leftarrow 1$ **to** $n-1$ **do**
4. Let π_k be an empty set.
5. **while** π_{k-1} is not empty **do**
6. Remove a member of π_{k-1} and denote it by $C^{k-1}_{(n-k):(n-1)}$ having N rows and k columns.
7. Determine those 2^{n-1} pairs of rows in $C^{k-1}_{(n-k):(n-1)}$ such that the row labels $i = (i_1 i_2 \dots i_n)$ and $j = (j_1 j_2 \dots j_n)$ of each pair are identical in their binary representation except that bits i_{k+1} and j_{k+1} are different.
8. Generate all those $2^{N/2}$ balanced matrices by swapping/unswapping the *contents* of each pair's rows of $C^{k-1}_{(n-k):(n-1)}$ independently of the other pairs of rows. (Note that the row labels are never swapped). Denote each generated matrix by $C^{k}_{(n-k):(n-1)}$.
9. Insert all the generated matrices into π_k.
10. **endwhile**
11. **if** $k < n-1$ **then**
12. Append (or place) column r_{n-k-1} of $R_{1:(n-1)}$ to the left of each member of π_k, so that r_{n-k-1} now becomes its first column.
13. Insert column r_{n-k-1} of $R_{1:(n-1)}$ into column f_{n-k-1} of $F_{1:(n-1)}$.
14. **endif**
15. **endfor**
16. Let π be identical to π_{n-1}.

Comment: π_{n-1} consists of all matrices fitting $F_{1:(n-1)}$.

end

FIG. 4.4. *Algorithm GENERATE-π.*

of the contents of the other pairs of rows, any matrix of π_1 generates new 2^8 matrices. In line 12 of GENERATE-π, column r_1 is placed to the left of each and every matrix of π_2. In the *third step*, each of the following eight sets contains the row labels of a pair of any matrix of π_2: $\{0,1\}$, $\{2,3\}$, $\{4,5\}$, $\{6,7\}$, $\{8,9\}$, $\{10,11\}$, $\{12,13\}$, and $\{14,15\}$. Each matrix of π_2 generates new 2^8 matrices fitting $F_{1:3}$. All these new generated matrices become the members of π_3.

LEMMA 4.4. *A matrix of order $N \times (n-1)$ fits $F_{1:(n-1)}$ if and only if the matrix is in the set π_{n-1} generated in algorithm GENERATE-π.*

Proof. Let $x_{1:(n-1)} = (x_1 x_2 \dots x_{n-1})$ and $y_{1:(n-1)} = (y_1 y_2 \dots y_{n-1})$ denote the

contents of rows $i = (i_1 i_2 \dots i_n)$ and $j = (j_1 j_2 \dots j_n)$, respectively, of $R_{1:(n-1)}$, where i and j are row labels. Because the row labels of $F_{1:(n-1)}$ and any matrix of π_k, $1 \le k \le n-1$, are in ascending order from 0 to $N-1$, the binary representation of the row labels form the identity permutation matrix $I_{1:n}$. By Definition 2.1, $I_{1:n}$ is the matrix whose $(k+1)$th column is the $(n-k)$th column of $R_{1:(n-1)}$. Therefore, if the row labels i and j are identical in their binary representation except that their $(k+1)$th bits (i.e., bits i_{k+1} and j_{k+1}) are different, then row contents $x_{1:(n-1)}$ and $y_{1:(n-1)}$ are identical except that their $(n-k)$th bits (i.e., bits x_{n-k} and y_{n-k}) are different.

Because the lemma considers the members of π_{n-1}, we can assume that all columns of $R_{1:(n-1)}$ are inserted into $F_{1:(n-1)}$. (Note that when columns of $R_{1:(n-1)}$ are inserted one by one into $F_{1:(n-1)}$, $R_{1:(n-1)}$ is inserted into $F_{1:(n-1)}$ eventually.) Recall from Lemma 4.3 that $R_{1:(n-1)}$ fits $F_{1:(n-1)}$. When the contents $x_{1:(n-1)}$ and $y_{1:(n-1)}$ of a pair of $C^{k-1}_{(n-k):(n-1)}$ are swapped (without swapping the row labels), bits x_{n-m} and y_{n-m} change their positions *within the same block* of column $n-m$ of $F_{1:(n-1)}$, for each and every m, $1 \le m \le k$. Therefore, when the contents $x_{1:(n-1)}$ and $y_{1:(n-1)}$ of a pair of $C^{k-1}_{(n-k):(n-1)}$ are swapped independently of the other pairs of rows, the difference between bits x_{n-m} and y_{n-m} does not prevent the generated matrix from fitting $F_{1:(n-1)}$. Thus, any $C^{n-1}_{1:(n-1)}$ of π_{n-1} fits $F_{1:(n-1)}$.

We will now show that any matrix that fits $F_{1:(n-1)}$ is contained in the set π_{n-1}. Let $A^{n-1}_{1:(n-1)}$ be an arbitrary matrix that fits $F_{1:(n-1)}$. To show that $A^{n-1}_{1:(n-1)}$ is contained in π_{n-1}, $A^{n-1}_{1:(n-1)}$ can be converted to $R_{1:(n-1)}$ by backtracking operations of algorithm GENERATE-π from step $n-1$ to step 1 as follows. In the first step, $A^{n-1}_{1:(n-1)}$ can be converted to a matrix $[r_1\ A^{n-2}_{2:(n-1)}]$ by swapping/unswapping the rows' contents as in line 8 of GENERATE-π for $k = n-1$. In the second step, $[r_1\ A^{n-2}_{2:(n-1)}]$ can be converted to a matrix $[r_1 r_2\ A^{n-3}_{3:(n-1)}]$ by swapping/unswapping the rows' contents as in line 8 of GENERATE-π for $k = n-2$. In the $(n-1)$th step, $[r_1 r_2 \dots r_{n-2}\ A^{1}_{(n-1):(n-1)}]$ can be converted to a matrix $[r_1 r_2 \dots r_{n-1}] = R_{1:(n-1)}$ by swapping/unswapping the rows' contents as in line 8 of GENERATE-π for $k = 1$. Thus $A^{n-1}_{1:(n-1)}$ can be generated by algorithm GENERATE-π by starting with r_{n-1} and then producing $A^{1}_{(n-1):(n-1)}$, $A^{2}_{(n-2):(n-1)}$, $\dots$, $A^{n-1}_{1:(n-1)}$ in order. Also, it is shown in the previous paragraph that any matrix of π_{n-1} fits $F_{1:(n-1)}$. Note that π_{n-1} consists of $2^{N(n-1)/2}$ matrices because each matrix of π_{k-1} in step k of GENERATE-π generates new $2^{N/2}$ matrices that become the members of π_k. Because it is shown in Lemma 3.4 that the maximum number of matrices fitting $F_{1:(n-1)}$ is $2^{N(n-1)/2}$, π_{n-1} consists precisely of those $2^{N(n-1)/2}$ matrices that fit $F_{1:(n-1)}$. □

Swapping and unswapping the contents of a pair's rows of a matrix are equivalent to routing the contents in two different ways through a switch of size (2×2) that can be set cross or straight independently of other switches, where swapping and unswapping correspond to setting the switch to cross and straight, respectively. Algorithm GENERATE-π can be easily implemented by an $(n-1)$-stage ISE network (denoted $ISE_{1:(n-1)}$) in which the label of output i is equal to the inverse shuffle-permutation $\gamma_{n-1}(i)$, and the inputs are labeled by γ_0 as usual in ascending order from 0 to $N-1$. An ISE with such input and output labels is shown in Figure 4.5 for $N = 16$. The $n-1$ stages of the ISE network are denoted by ISE_1, $ISE_2, \dots, ISE_{n-1}$ from the leftmost stage to the rightmost stage, respectively. Note that the label of input i of ISE_k is equal to $\gamma_{k-1}(i)$, which is defined in the beginning of section 3. Algorithm

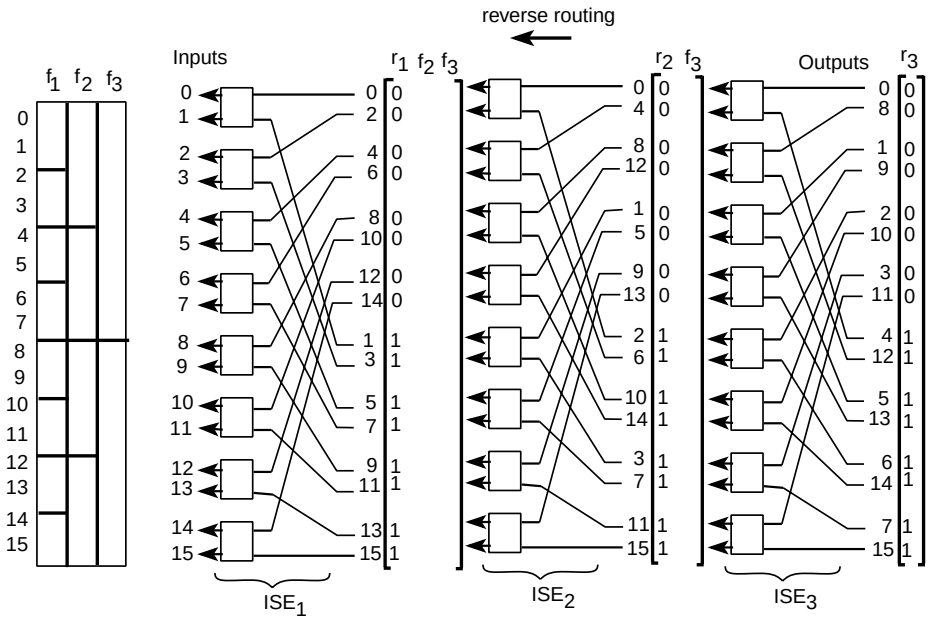


FIG. 4.5. *The inverse-shuffle exchange network* $ISE_{1:(n-1)}$ *for* $N = 16$. *The stages are labeled from left to right. As explained in section* 2, *input* i *of* ISE_j *is labeled by* $\gamma_{j-1}(i)$ *for all* i, $0 \le i \le N-1$. *Algorithm GENERATE-*π *is implemented as follows:* (1) *reverse-route column* r_3 *through* ISE_3 *in all switch settings;* (2) *insert* r_2 *to the left of any column vector that is obtained at the inputs of* ISE_3, *and reverse-route each of the resulting two-column matrices through* ISE_2 *in all switch settings;* (3) *insert* r_1 *to the left of any two-column matrix that was obtained at the inputs of* ISE_2, *and reverse-route each of the resulting three-column matrices through* ISE_1 *in all switch settings. Finally, all matrices that fit* $F^0_{1:3}$ *are obtained at the inputs side of* ISE_1.

GENERATE-π is implemented by $ISE_{1:(n-1)}$ as follows. In the first step, the entries of column r_{n-1} are *reverse-routed* through the stage ISE_{n-1} from its outputs to inputs in all possible $2^{N/2}$ settings of switches. In the second step, column r_{n-2} is first placed to the left of each generated column vector, and then the resulting matrix is reverse-routed through ISE_{n-2} in all possible $2^{N/2}$ different ways. In general, in the kth step, column r_{n-k} is first placed to the left of each generated matrix that is available at the outputs side of ISE_{n-k}, and then the resulting matrix is reverse-routed through ISE_{n-k} in all possible ways. Notice that those two entries of r_{n-k} that are reverse-routed through the same switch of ISE_{n-k} constitute the set $\{0, 1\}$, thereby forming a pair of rows mentioned in GENERATE-π. The pairs of row labels that are formed in step k of GENERATE-π are the same as the pairs of row labels formed by the switches of ISE_{n-k}. Thus the set of all those matrices that are generated by reverse-routing r_{n-1}, r_{n-2}, ..., r_1 through $ISE_{1:(n-1)}$ in the way described above is identical to the set π_{n-1}.

4.2. Algorithm GENERATE-ALL and rearrangeability of $RB_{1:(n-1)}SE_{1:n}$. Let S denote the set of all $N!$ distinct balanced matrices of order $N \times n$. Also, let T denote the set of all those balanced matrices of order $N \times (2n-1)$, each of which fits $F^0_{1:(2n-1)}$ (or denoted $F_{1:(2n-1)}$). We now aim to show that each member of S appears as the last n columns of at least one member of T. Specifically, we show

that for any balanced matrix, say, $B_{1:n}$, of S, there exists at least one $C^{n-1}_{1:(n-1)}$ fitting $F_{1:(n-1)}$ such that the matrix $[C^{n-1}_{1:(n-1)}\ B_{1:n}]$ is balanced. We propose another algorithm, called GENERATE-ALL, to generate all matrices fitting the frame $F_{1:(2n-1)}$ in $n-1$ steps. Each step has two objectives. The first objective is to keep guaranteeing that the submatrices formed by the last n columns of all matrices generated in each step constitute the set S, starting with the set S initially. The second objective is to gradually generate all those matrices fitting $F_{1:(n-1)}$ in the same way that algorithm GENERATE-π generates them and to guarantee that $[C^{n-1}_{1:(n-1)}\ B_{1:n}]$ is balanced.

Algorithm GENERATE-ALL is presented next, where the kth iteration of the *for* loop is referred to as *step* k. To show the steps of GENERATE-ALL for $N = 16$, an example is provided just after the algorithm.

Algorithm GENERATE-ALL.

Input: A set of all $N!$ distinct balanced matrices of order $N \times n$. An $(n-1)$-stage inverse-shuffle exchange $ISE_{1:(n-1)}$ network such that the label of input i of ISE_k is equal to $\gamma_{k-1}(i)$ and the label of output i of ISE_k is $\gamma_k(i)$.

Output: All those balanced matrices of order $N \times (2n-1)$ that fit $F_{1:(2n-1)}$ such that each balanced matrix of order $N \times n$ appears in the last n columns of at least one of them. The sets α_{n-1} and β_{n-1} are generated. α_{n-1} contains all balanced matrices that fit $F_{1:(2n-1)}$. β_{n-1} contains every distinct submatrix that is formed by the last n columns of each member of α_{n-1}.

begin

1. Let α_0 denote the set of all $N!$ distinct balanced matrices of order $N \times n$ such that rows are labeled by the inverse-shuffle permutation γ_{n-1}. A member of α_0 is denoted by $A^0_{n:(2n-1)}$ whose columns are labeled by $n,\ n+1, \ldots, 2n-1$ from left to right. Let β_0 be identical to α_0.
2. Let α_k and β_k be empty sets for each and every k, $1 \le k \le n-1$.
3. **for** $k \leftarrow 1$ **to** $n-1$ **do**
4. Let $A^{k-1}_{(n-k+1):(2n-1)}$ denote a member of α_{k-1}. Insert column r_{n-k} to the left of each $A^{k-1}_{(n-k+1):(2n-1)}$ such that the ith entry of r_{n-k} is placed to the left of the ith row of $A^{k-1}_{(n-k+1):(2n-1)}$. If the resulting matrix $[r_{n-k}\ A^{k-1}_{(n-k+1):(2n-1)}]$ is balanced, then keep it in α_{k-1}; otherwise, remove it from α_{k-1}.
5. **while** α_{k-1} is not empty **do**
6. Pick up a member $[r_{n-k}\ A^{k-1}_{(n-k+1):(2n-1)}]$ of α_{k-1} and remove it from α_{k-1}.
7. Generate $2^{N/2}$ matrices by reverse-routing the *contents* of all rows of $[r_{n-k}\ A^{k-1}_{(n-k+1):(2n-1)}]$ through in all $2^{N/2}$ switch settings of stage ISE_{n-k}. Note that in reverse-routing, the matrix row with label $\gamma_{n-k}(i)$ is reverse-routed through output i of stage ISE_{n-k}.
8. Insert each generated matrix, denoted $A^k_{(n-k):(2n-1)}$, into α_k. Also, insert the submatrix $A^k_{n:(2n-1)}$ of each $A^k_{(n-k):(2n-1)}$ into β_k if the submatrix is not already a member of β_k.
9. **endwhile**
10. **endfor**

end

Before introducing formal proofs related to the identification and combinatorial properties of the matrices generated by algorithm GENERATE-ALL, an example is presented next to show how algorithm GENERATE-ALL generates matrices.

Example 4.3. Let $N = 16$ and $n = 4$. All those matrices fitting $F^0_{1:7}$ are generated in three steps. To illustrate how GENERATE-ALL works, Figures 4.6, 4.7, and 4.8,

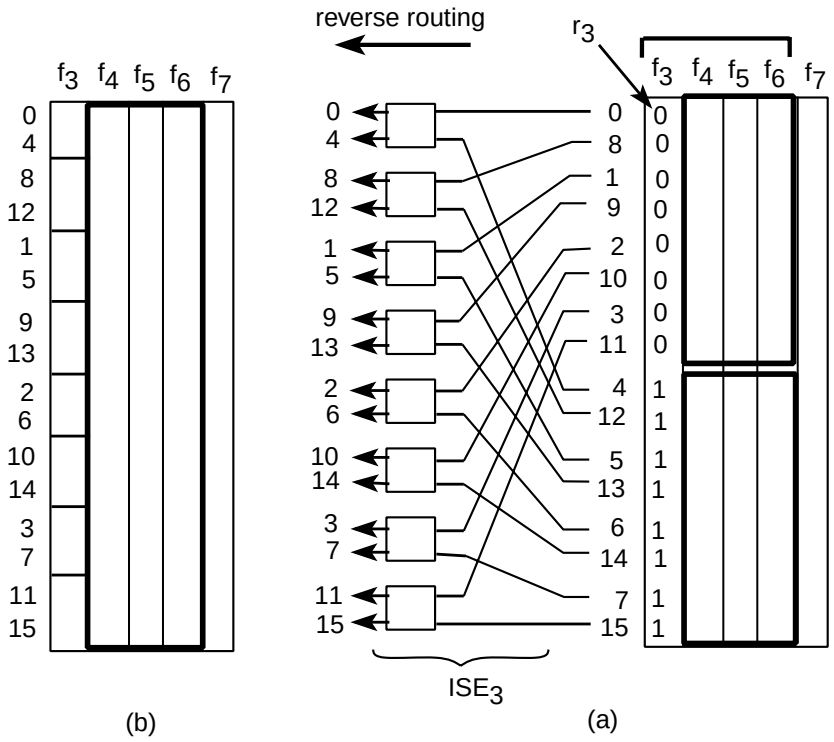


FIG. 4.6. *The implementation of the first step of GENERATE-ALL for* $N = 16$.

corresponding to steps 1, 2, and 3, respectively, will be discussed in this example. In step k, stage ISE_{n-k} is used.

Initially, α_0 contains all 16! balanced matrices of order 16×4. In step 1, column r_3 is placed to the left of every member of α_0 such that the ith entry of r_3 is placed to the left of the ith row of each member of α_0, as shown in Figure 4.6(a). If the resulting matrix is balanced, then it remains in α_0; otherwise, it is removed from α_0. In particular, every matrix that fits $F^2_{4:7}$ remains balanced when r_3 is inserted to the left; thus all these matrices remain in α_0. ($F^2_{4:7}$ consists of a pile of two universal frames each having $N/2$ rows and $n-1$ columns, followed by a single block of N rows, as shown in Figure 4.6 for $N = 16$.) Then each member of the latest α_0 is reverse-routed through ISE_3 in all $2^{16/2}$ different ways, and the matrices that are obtained at the inputs side of ISE_3 constitute the set α_1. Each member of α_1 fits the frame shown in Figure 4.6(b). All submatrices of order $N \times n$ that are formed by the last n columns of every member of α_1 are inserted into β_1. It will be shown later that β_1 contains all $N!$ balanced matrices of order $N \times n$. Note that each of the following eight sets contains the input/output labels of a switch of ISE_3: $\{0,4\}$, $\{8,12\}$, $\{1,5\}$, $\{9,13\}$, $\{2,6\}$, $\{10,14\}$, $\{3,7\}$, and $\{11,15\}$. These eight sets are also the same as the sets of eight row pairs of column r_3 that are formed in algorithm GENERATE-π.

In step 2, the frames and matrices of α_1 that are obtained at the end of step 1 are now located at the outputs side of ISE_2, as shown in Figure 4.7(a). Column r_2 is

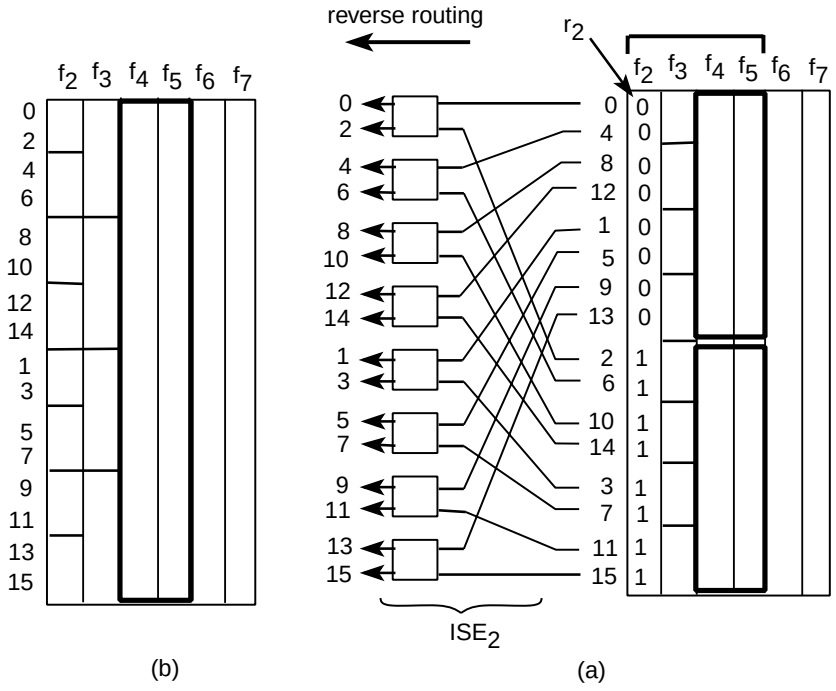


FIG. 4.7. *The implementation of the second step of GENERATE-ALL for* $N = 16$.

placed to the left of every member of α_1. If the resulting matrix is balanced, then it remains in α_1; otherwise, it is removed from α_1. Then each member of the latest α_1 is reverse-routed through ISE_2 in all $2^{16/2}$ different ways, and the matrices that are obtained at the inputs side of ISE_2 constitute the set α_2. Each member of α_2 fits the frame shown in Figure 4.7(b). All submatrices that are formed by the last n columns of every member of α_2 are inserted into β_2. It will be shown later that β_2 contains all $N!$ balanced matrices of order $N \times n$. Note that each of the following eight sets contains the input/output labels of a switch of ISE_2: $\{0, 2\}$, $\{4, 6\}$, $\{8, 10\}$, $\{12, 14\}$, $\{1, 3\}$, $\{5, 7\}$, $\{9, 11\}$, and $\{13, 15\}$. These eight sets are the same as the sets of eight pairs of column r_2 that are formed in algorithm GENERATE-π.

In step 3, column r_1 is placed to the left of each member of α_2, as shown in Figure 4.8(a). If the resulting matrix is balanced, then it remains in α_2; otherwise, it is removed from α_2. Then each member of the latest α_2 is reverse-routed through ISE_1 in all $2^{16/2}$ different ways, and the matrices that are obtained at the inputs side of ISE_1 constitute the set α_3. Each member of α_3 fits the frame $F^0_{1:7}$ shown in Figure 4.8(b). All submatrices that are formed by the last n columns of every member of α_3 are inserted into β_3. It will be shown later that β_3 contains all $N!$ balanced matrices of order $N \times n$. Note that each of the following eight sets contains the input/output labels of a switch of ISE_1: $\{0, 1\}$, $\{2, 3\}$, $\{4, 5\}$, $\{6, 7\}$, $\{8, 9\}$, $\{10, 11\}$, $\{12, 13\}$, and $\{14, 15\}$. These eight sets are the same as the sets of eight pairs of column r_1 that are formed in algorithm GENERATE-π.

Lemma 4.5 shows that $A^{n-1}_{1:(2n-1)}$ fits $F_{1:(2n-1)}$; that is, $A^{n-1}_{1:(2n-1)}$ is balanced, and

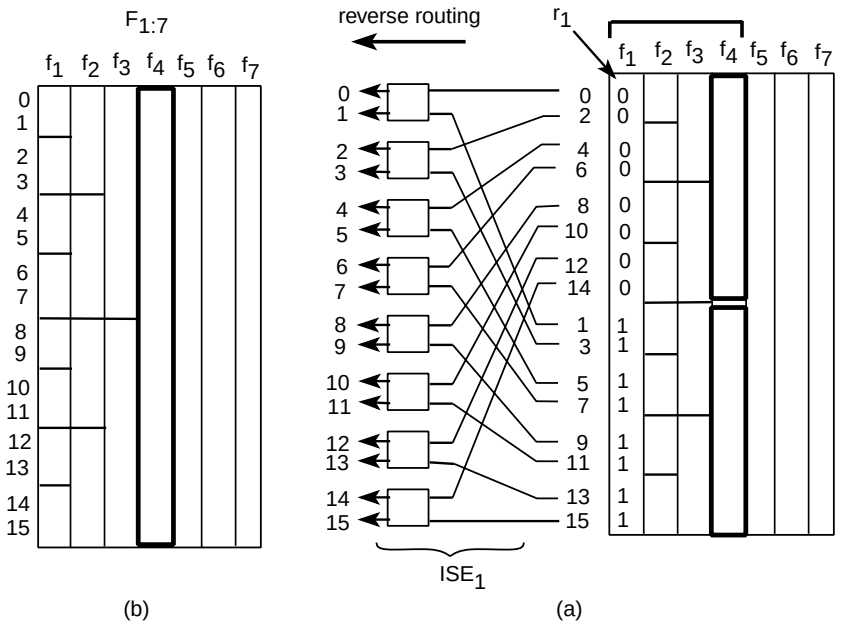


FIG. 4.8. *The implementation of the third step of GENERATE-ALL for* $N = 16$.

$A^{n-1}_{1:(n-1)}$ fits $F_{1:(n-1)}$. Theorem 4.6 proves that β_k contains any balanced matrix of order $N \times n$.

LEMMA 4.5. *Any member* $A^{n-1}_{1:(2n-1)}$ *of the set* α_{n-1} *fits* $F_{1:(2n-1)}$.

Proof. Let $M_{1:(2n-1)}$ be a matrix of order $N \times (2n-1)$ whose first $n-1$ columns (i.e., $M_{1:(n-1)}$) fit $F_{1:(n-1)}$. By the definition of fit (Definition 3.3), in order for a matrix to fit a frame, each p-column block of the frame is required to contain a balanced matrix when row i of the matrix is placed into row i of the frame. Note that each column of the subframe $F_{n:(2n-1)}$ of $F_{1:(2n-1)}$ is a single block of N entries. Therefore, when $F_{n:(2n-1)}$ is appended to the right of $F_{1:(n-1)}$ to form $F_{1:(2n-1)}$, the columns of $F_{n:(2n-1)}$ require the above matrix $M_{1:(2n-1)}$ to be balanced only. (Recall from Definition 2.6 that a matrix having more than n columns is balanced if each and every n consecutive column forms a balanced matrix.) This implies that, in order for a matrix $A^{n-1}_{1:(2n-1)}$ to fit $F_{1:(2n-1)}$, the following two conditions are necessary and sufficient: (1) $A^{n-1}_{1:(n-1)}$ fits $F_{1:(n-1)}$, and (2) $A^{n-1}_{1:(2n-1)}$ is balanced.

We shall first show that the first condition holds. Note that input labels of a switch of ISE_{n-k} are the same as the row labels of a pair of $C^{k-1}_{(n-k):(n-1)}$ generated in step k of GENERATE-π. Also, reverse-routing the contents of a pair's rows through a switch in two different ways independently of other switches is equivalent to swapping/unswapping the contents of the pair's rows independently of other pairs. In addition, the first $n-1$ columns of any $A^{n-1}_{1:(2n-1)}$ are generated in $n-1$ steps by first inserting column r_{n-k} of the reverse permutation matrix into f_{n-k} in step k and then reverse-routing the rows of the existing matrix through ISE_{n-k}. This implies that any submatrix $A^{n-1}_{1:(n-1)}$ is generated in GENERATE-ALL in the same way as

any $C^{n-1}_{1:(n-1)}$ is generated in GENERATE-π. Hence it follows from Lemma 4.4 that $A^{n-1}_{1:(n-1)}$ fits $F^0_{1:(n-1)}$. Thus the first condition holds.

When the rows of a balanced matrix are swapped/unswapped arbitrarily, the resulting matrix is still balanced. After inserting column r_{n-k} to the left of each member of α_{k-1} in the beginning of step k, those resulting matrices that happen to be unbalanced are removed from α_{k-1}. Thus only balanced matrices are reverse-routed through ISE_{n-k} in step k. Reverse-routing a matrix just swaps/unswaps the contents of rows, which does not make a balanced matrix unbalanced. Therefore, all the matrices of α_k that are obtained at the inputs side of ISE_{n-k} in step k are balanced. Hence any $A^{n-1}_{1:(2n-1)}$ is balanced, and the second condition holds as well. Therefore, $A^{n-1}_{1:(2n-1)}$ fits $F_{1:(2n-1)}$. □

In the beginning of every step of algorithm GENERATE-ALL, β_{k-1} contains every balanced matrix of order $N \times n$. However, then r_{n-k} is inserted to the left of each member of α_{k-1}, and the resulting *unbalanced* matrices are removed from α_{k-1}. Because the operation of inserting column r_{n-k} to the left of each member of α_{k-1} starts with step 1, Theorem 4.6 treats step 1 as a "special" case. The proof of step 1 is "easier" because it follows directly from Corollary 4.2. However, the other steps make use of the results of both Corollary 4.2 and algorithm GENERATE-π.

THEOREM 4.6. *The set β_k contains all $N!$ balanced matrices of order $N \times n$ for each k, $1 \le k \le n-1$.*

Proof. β_k is the set of all those balanced matrices of order $N \times n$, each of which appears in the last n columns of at least one member of α_k. Thus a member of β_k is denoted by the submatrix $A^k_{n:(2n-1)}$.

Proof of Step 1. Let $k = 1$. Initially, α_0 denotes a set containing any balanced matrix of order $N \times n$. Specifically, α_0 contains any balanced matrix fitting the universal frame $F^{n-1}_{n:(2n-1);\gamma_{n-1}}$. When r_{n-1} is inserted to the left of each member $A^0_{n:(2n-1)}$ of α_0, we examine whether any submatrix $[r_{n-1}\ A^0_{n:(2n-2)}]$ is a balanced matrix. Those matrices $A^0_{n:(2n-1)}$ for which $[r_{n-1}\ A^0_{n:(2n-2)}]$ is not balanced are removed from α_0. Now let us identify those matrices that are not removed from α_0. Note that the upper 2^{n-1} entries of r_{n-1} are 0's, and the lower 2^{n-1} entries are 1's. The 2^{n-1} 0's of r_{n-1} require that any $(n-1)$-bit row must appear only *once* in the upper 2^{n-1} rows of the submatrix $A^0_{n:(2n-2)}$ so that $[r_{n-1}\ A^0_{n:(2n-2)}]$ becomes a balanced matrix. Likewise, the 2^{n-1} 1's of r_{n-1} require that any $(n-1)$-bit row must appear only *once* in the lower 2^{n-1} rows of the submatrix $A^0_{n:(2n-2)}$ so that $[r_{n-1}\ A^0_{n:(2n-2)}]$ becomes a balanced matrix. Because this is the only constraint imposed by r_{n-1}, any balanced matrix of order $2^{n-1} \times (n-1)$ can occur in the upper 2^{n-1} rows of the submatrix $A^0_{n:(2n-2)}$ as well as in the lower 2^{n-1} rows of the submatrix $A^0_{n:(2n-2)}$. This amounts to saying that the latest α_0 contains any matrix fitting $F^{n-2}_{n:(2n-2);\gamma_{n-1}}$ which consists of a pile of two universal frames, denoted $UF_{n:(2n-2)}$ and $LF_{n:(2n-2)}$, each having 2^{n-1} rows and $n-1$ columns. When each matrix of the latest α_0 is reverse-routed through ISE_{n-1} in all $2^{N/2}$ different ways, all matrices that are obtained at the inputs side of ISE_{n-1} constitute the set α_1. In the reverse-routing of all matrices, note that a distinct row label of $UF_{n:(2n-2)}$ and a distinct row label of $LF_{n:(2n-2)}$ are paired and connected to the outputs of the same switch. Therefore, it follows from Corollary 4.2 that $F^{n-2}_{n:(2n-2);\gamma_{n-1}}$ is converted to the universal frame $F^{n-1}_{n:(2n-2);\gamma_{n-2}}$, and, therefore, any submatrix fitting $F^{n-1}_{n:(2n-1);\gamma_{n-2}}$ is obtained at the inputs side of ISE_{n-1}. Therefore, β_1 contains every balanced matrix

of order $N \times n$. Thus the theorem holds for $k = 1$.

Proof of Steps 2 *to* $n-1$. Let $2 \le k \le n-1$. It is shown above that the theorem holds for β_1. Because steps 2 to $n-1$ are implemented successively, we assume that the theorem holds for β_{k-1} and show that it holds for β_k as well. When r_{n-k} is inserted to the left of each member $A^{k-1}_{(n-k+1):(2n-1)}$ of α_{k-1} in the beginning of step k (line 4 of algorithm GENERATE-ALL), the submatrix $[r_{n-k}\ A^{k-1}_{(n-k+1):(n-1)}]$ automatically becomes balanced because $A^{k-1}_{(n-k+1):(n-1)}$ has been constructed by reverse-routing columns $r_{n-1},\ r_{n-2}, \ldots, r_{n-k+1}$ through $ISE_{n-1},\ ISE_{n-2},\ \ldots,\ ISE_{n-k+1}$ in the way explained earlier for the implementation of algorithm GENERATE-π. ($A^{k-1}_{(n-k+1):(n-1)}$ fits $F^0_{1:(k-1);\gamma_{n-k}}$, and eventually $A^{n-1}_{1:(n-1)}$ fits $F^0_{1:(n-1)}$ as shown in the proof of Lemma 4.5.) Thus, when r_{n-k} is inserted to the left of each $A^{k-1}_{(n-k+1):(2n-1)}$, $[r_{n-k}\ A^{k-1}_{(n-k+1):(n-1)}]$ is always balanced, and, therefore, the set of submatrices $A^{k-1}_{(n-k+1):(n-1)}$ remains the same when column r_{n-k} is inserted. Thus r_{n-k} can lead only the set of submatrices $A^{k-1}_{n:(2n-k-1)}$ to be changed.

In order for the matrix $[r_{n-k}\ A^{k-1}_{(n-k+1):(2n-k-1)}]$ to be balanced, the following must be satisfied: (1) any $(n-1)$-bit row must appear *once* in the upper 2^{n-1} rows of any submatrix $A^{k-1}_{(n-k+1):(2n-k-1)}$ because the upper 2^{n-1} entries of r_{n-k} are all 0's, and (2) any $(n-1)$-bit row must appear *once* in the lower 2^{n-1} rows of any submatrix $A^{k-1}_{(n-k+1):(2n-k-1)}$ because the lower 2^{n-1} entries of r_{n-k} are all 1's. Thus r_{n-k} requires both the upper 2^{n-1} rows and the lower 2^{n-1} rows of any submatrix $A^{k-1}_{n:(2n-k-1)}$ to have a balanced matrix of order $2^{n-1} \times (n-k)$. However, (1) prior to insertion of r_{n-k}, the set of submatrices $A^{k-1}_{n:(2n-k-1)}$ contained every balanced matrix of order $N \times (n-k)$ because the set β_{k-1} contains any balanced matrix of order $N \times n$, and (2) it is shown above that r_{n-k} affects only the set of submatrices $A^{k-1}_{n:(2n-k-1)}$. Therefore, after inserting column r_{n-k}, any balanced matrix of order $2^{n-1} \times (n-k)$ can occur in the upper 2^{n-1} rows of the submatrix $A^{k-1}_{n:(2n-k-1)}$ as well as in the lower 2^{n-1} rows of the submatrix $A^{k-1}_{n:(2n-k-1)}$. This amounts to saying that the set of all submatrices $A^{k-1}_{n:(2n-k-1)}$ contains any matrix fitting a frame $F^{n-2}_{n:(2n-k-1);\gamma_{n-k}}$, which consists of a pile of two universal frames, denoted $UF_{n:(2n-k-1)}$ and $LF_{n:(2n-k-1)}$, each having $N/2$ rows and $n-k$ columns. When each matrix of the latest α_{k-1} is reverse-routed through ISE_{n-k} in all $2^{N/2}$ different ways, all matrices that are obtained at the inputs side of ISE_{n-1} constitute the set α_k. In the reverse-routing of all matrices, note that a distinct row label of $UF_{n:(2n-k-1)}$ and a distinct row label of $LF_{n:(2n-k-1)}$ are paired and connected to the outputs of the same switch. Therefore, it follows from Corollary 4.2 that $F^{n-2}_{n:(2n-k-1);\gamma_{n-k}}$ is converted to the universal frame $F^{n-1}_{n:(2n-k-1);\gamma_{n-k-1}}$ and, therefore, any submatrix fitting the universal frame $F^{n-1}_{n:(2n-1);\gamma_{n-2}}$ is obtained at the inputs side of ISE_{n-k}. Therefore, β_k contains every balanced matrix of order $N \times n$. Thus the theorem holds for any k. □

The purpose of obtaining all the results so far in this section is to help prove that $RB_{1:(n-1)}SE_{1:n}$ is a rearrangeable network.

THEOREM 4.7. *$RB_{1:(n-1)}SE_{1:n}$ is a rearrangeable network.*

Proof. Lemma 4.5 has shown that any $A^{n-1}_{1:(2n-1)}$ of α_{n-1} fits $F_{1:(2n-1)}$. Therefore, it follows from Theorem 3.7 that any $A^{n-1}_{1:(2n-1)}$ passes the network $RB_{1:(n-1)}SE_{1:n}$, and the network $RB_{1:(n-1)}SE_{1:n}$ realizes the permutation represented by the subma-

trix $A^{n-1}_{n:(2n-1)}$. Because $A^{n-1}_{n:(2n-1)}$ is a member of β_{n-1} and Theorem 4.6 has shown that β_{n-1} contains every balanced matrix of order $N \times n$, the network realizes any permutation. Thus $RB_{1:(n-1)}SE_{1:n}$ is rearrangeable. □

5. Rearrangeability of SE networks. In this section, the rearrangeability of $SE_{1:(2n-1)}$ is proven in Theorem 5.2. As a corollary to this result, it is shown in Corollary 5.3 that two passes through an Omega network are sufficient to realize any permutation. Before proceeding to the proofs of these theorems, a preliminary result concerning the functional equivalence of $RB_{1:n}$ and $SE_{1:n}$ is introduced.

Two INs are *functionally equivalent* if they realize the same set of permutations, whereas two INs are *topologically equivalent* if the graphs corresponding to their topologies are isomorphic. (Two graphs are isomorphic if there exists a bijection from the vertices of one of these graphs into the vertices of the other one such that the relationship of adjacency is preserved [5].) Given a set of topologically equivalent networks, one can rename the inputs and/or outputs of one of these networks to simulate any other network in the set. In [13, 17], it is shown that $RB_{1:n}$ and $SE_{1:n}$ are topologically equivalent. Lemma 5.1 shows that they can also be made functionally equivalent by relabeling their inputs. Its proof is provided in the appendix.

LEMMA 5.1. *$RB_{1:n}$ and $SE_{1:n}$ are functionally equivalent if the ith input of one of them is relabeled by the value of $R_{N\times n}(i)$, where $0 \le i \le N-1$.* □

THEOREM 5.2. *The $(2n-1)$-stage SE network, $SE_{1:(2n-1)}$, is rearrangeable.*

Proof. Let IP_a be an interconnection pattern such that its inputs are labeled from 0 to $N-1$ in ascending order, and the label of its ith output is equal to the value of $R_{N\times n}(i)$, where $0 \le i \le N-1$. So IP_a implements a permutation, say, r, that sends (or maps) the input i to the output whose value equals $R_{N\times n}(i)$. By Lemma 5.1, $IP_aRB_{1:n}$ is functionally equivalent to $SE_{1:n}$. Therefore, $SE_{1:(2n-1)}$ is functionally equivalent to $IP_aRB_{1:n}SE_{1:(n-1)}$. Also, because $RB_{1:n}SE_{1:(n-1)}$ is identical to $RB_{1:(n-1)}SE_{1:n}$, $SE_{1:(2n-1)}$ is functionally equivalent to $IP_aRB_{1:(n-1)}SE_{1:n}$. Let S and T be the set of permutations realized by $SE_{1:(2n-1)}$ and $RB_{1:(n-1)}SE_{1:n}$ networks, respectively. It follows that $S = r \times T$. Because $RB_{1:(n-1)}SE_{1:n}$ is rearrangeable by Theorem 4.7, T contains $N!$ distinct permutations. When each of the $N!$ distinct permutations is multiplied by the same permutation r, the set of $N!$ permutations is mapped onto itself. Therefore, $SE_{1:(2n-1)}$ is rearrangeable. □

COROLLARY 5.3. *For $k \ge 0$, $SE_{1:(2n+k)}$ is rearrangeable.*

Proof. Assume that all the switches of the subnetwork $SE_{2n:(2n+k)}$ are always set straight. This leads $SE_{2n:(2n+k)}$ to be functionally equivalent to an interconnection pattern, and hence it always realizes the same permutation, say, b. Thus the set of permutations realized by $SE_{1:(2n+k)}$ is obtained by multiplying each and every permutation realized by $SE_{1:(2n-1)}$ by b. By Theorem 5.2, the cardinality of the set of the permutations realized by $SE_{1:(2n-1)}$ is $N!$. Because multiplying each and every member of the set of $N!$ permutations by the same permutation b results in $N!$ distinct permutations, $SE_{1:(2n+k)}$ is rearrangeable. □

Corollary 5.3 implies that two passes through an Omega network are sufficient to realize any permutation due to the fact that the network obtained by cascading two Omega networks in serial is the same as $SE_{1:2n}$. This implies that a network consisting of two cascaded butterflies is, in general, rearrangeable.

6. Conclusions. This paper has proven that the $(2n-1)$-stage SE network with 2^n inputs and 2^n outputs is a rearrangeable network; that is, it can realize each and every one of $2^n!$ permutations in a single pass through the network. The paper has also

shown that two passes through the Omega network are sufficient to implement any permutation. In obtaining these results, permutations realized by some multistage INs are identified using the notion of balanced matrices and frames. We have shown that frames are a very useful tool in identifying and characterizing permutations realized by multistage INs.

Appendix.

Proof of Theorem 3.5. Recall that a matrix has $N = 2^n$ rows, unless otherwise stated. ($\Rightarrow$) It is shown that (1) if $D_{1:k}$ fits $F_{1:k}$, then $D_{1:k}$ passes $RB_{1:k}$, and (2) $RB_{1:k}$ sends its ith input to its jth output, where j is equal to the sum of $(\lfloor i/2^k \rfloor \times 2^k)$ and the value of $D_{1:k}(i)$. The proof is by induction on k.

Basis step. Let $k = 1$. By definition, f_1 contains 2^{n-1} blocks, each of size 2. By induction hypothesis, $D_{1:k}$ fits $F_{1:k}$, which implies that the leftmost column d_1 fits f_1. Therefore, the $2r$th and $(2r + 1)$th entries of d_1 constitute the set $\{0, 1\}$, where $0 \le r \le 2^{n-1} - 1$. Hence, when the $2r$th and $(2r + 1)$th entries of d_1 are used as the control bits to set the rth SB of $RB_{1:1}$, no conflict occurs, and $RB_{1:1}$ sends its ith input to its jth output, where j is equal to the sum of $(\lfloor i/2 \rfloor \times 2)$ and the value of the ith entry of d_1, where $0 \le i \le N - 1$.

Induction step. Let $2 \le k \le n$. Assuming that the theorem holds for $k - 1$ in the ($\Rightarrow$) direction, it is shown next that the theorem also holds for k in the ($\Rightarrow$) direction.

Note that $RB_{1:m}$ $(k - 1 \le m \le k)$ consists of a pile of 2^{n-m} copies of $RB_{2^m \times m}$. Let RB^a and RB^b denote the top two $RB_{2^{k-1} \times (k-1)}$'s of $RB_{1:(k-1)}$. Also, let $RB_{2^k \times k}$ denote the topmost $RB_{2^k \times k}$ of $RB_{1:k}$. By induction hypothesis, $D_{1:(k-1)}$ passes $RB_{1:(k-1)}$. This implies that, for $0 \le p \le 2^{k-1} - 1$, the first $(k - 1)$ entries of the row that is sent to the pth output of RB^a are the same as the first $(k - 1)$ entries of the row that is sent to the pth output of RB^b. Also, if $D_{1:k}$ fits $F_{1:k}$, the kth entries of those two rows sent to the pth outputs of RB^a and RB^b constitute the set $0, 1$. Moreover, those two rows enter the pth SB of the last SE stage of $RB_{2^k \times k}$, and their kth entries are the control bits of the SB. Hence no conflict occurs in the switches of the last stage of $RB_{2^k \times k}$, and $RB_{2^k \times k}$ sends its hth input to the output whose value is equal to the contents of $D_{1:k}(h)$, where $0 \le h \le 2^k - 1$. Because this holds for every $RB_{2^k \times k}$ of $RB_{1:k}$, the theorem also holds for k in the ($\Rightarrow$) direction.

($\Leftarrow$) It is shown that, if $D_{1:k}$ passes $RB_{1:k}$, then $D_{1:k}$ fits $F_{1:k}$. The proof is by induction on k.

Basis step. Let $k = 1$. The fact that d_1 passes RB_1 implies that no conflict occurs in the SBs of RB_1 when the ith entry of d_1 is used as the control bit for the ith input of RB_1 in setting its rth SB. Because the control bits of the rth SB of RB_1 constitute the set $\{0, 1\}$ and fit the rth block of f_1, d_1 fits f_1.

Induction step. Let $2 \le k \le n$. Assuming that the theorem holds for $k - 1$ in the ($\Leftarrow$) direction, it is shown next that the theorem also holds for k in the ($\Leftarrow$) direction.

As explained above, the rows that are sent to the pth outputs of RB^a and RB^b enter the pth SB of the last SE stage of $RB_{2^k \times k}$, and their kth entries are used as the control bits for the SB. So, if $D_{1:k}$ passes $RB_{1:k}$, then the kth entries of those two rows of $D_{1:k}$ that enter an SB of $RB_{1:k}$ must constitute the set $\{0, 1\}$ to avoid having a conflict in the SB. Therefore, $D_{1:k}$ fits $F_{1:k}$. □

Proof of Theorem 3.7. Because the last stage of $RB_{1:n}$ is an SE stage, the network $RB_{1:n}SE_{1:(m-1)}$ is the same as the network $RB_{1:(n-1)}SE_{1:m}$. Therefore, the theorem can be equivalently restated by substituting $RB_{1:n}SE_{1:(m-1)}$ for $RB_{1:(n-1)}SE_{1:m}$ as follows: a balanced matrix $D_{1:(n+m-1)}$, $m \ge 1$, fits the frame $F_{1:(n+m-1)}$ if and only

if $D_{1:(n+m-1)}$ passes the network $RB_{1:n}SE_{1:(m-1)}$, where $SE_{1:(m-1)}$ is assumed to be nil if $m = 1$. Moreover, $RB_{1:n}SE_{1:(m-1)}$ realizes the permutation represented by the last n columns of $D_{1:(n+m-1)}$, denoted by $D_{m:(n+m-1)}$. This is proven next.

($\Rightarrow$) It is shown that if $D_{1:(n+m-1)}$ fits $F_{1:(n+m-1)}$, then $D_{1:(n+m-1)}$ passes $RB_{1:n}SE_{1:(m-1)}$, and the permutation represented in binary by $D_{m:(n+m-1)}$ is realized by $RB_{1:n}SE_{1:(m-1)}$.

Because $D_{1:(n+m-1)}$ fits $F_{1:(n+m-1)}$ by hypothesis, $D_{1:n}$ fits $F_{1:n}$. Assume $RB_{1:n}$ maps the matrix $D_{1:(n+m-1)}$ located at the inputs side into the matrix $D^*_{1:(n+m-1)}$ located at the outputs side when $D_{1:(n+m-1)}(i)$, $0 \le i \le N-1$, is used as the routing tag for the ith input of $RB_{1:n}$. Theorem 3.5 has shown that any balanced matrix $D_{1:n}$ fitting the frame $F_{1:n}$ passes the network $RB_{1:n}$. So $RB_{1:n}$ maps any $D_{1:n}$ fitting the frame $F_{1:n}$ to $I_{1:n}$. This implies that, when $D_{1:(n+m-1)}(i)$ is used as the routing tag for the ith input of $RB_{1:n}SE_{1:(m-1)}$, the submatrix $D^*_{1:n}$ of $D^*_{1:(n+m-1)}$ is the same as the identity permutation matrix $I_{1:n}$. Therefore, $D^*_{1:(n+m-1)}$ is equal to the balanced matrix $[I_{1:n}\ D^*_{(n+1):(n+m-1)}]$. By Lemma 3.6, $SE_{1:(m-1)}$ realizes the permutation represented by $D^*_{m:(n+m-1)}$, and no conflict occurs in the SBs of $SE_{1:(m-1)}$ when $D^*_{(n+1):(n+m-1)}(i)$ is used as the routing tag for the ith input of $SE_{1:(m-1)}$. Therefore, $D_{1:(n+m-1)}$ passes $RB_{1:n}SE_{1:(m-1)}$. Also, if the entries of $D_{1:(n+m-1)}(i)$ are denoted in binary by $(x^i_1 x^i_2 \dots x^i_n \dots x^i_{n+m-1})$, then $RB_{1:n}SE_{1:(m-1)}$ sends $D_{1:(n+m-1)}(i)$ to the output whose value equals $(x^i_m x^i_{m+1} \dots x^i_{n+m-1})$. This means that the permutation represented by $D_{m:(n+m-1)}$ is realized by $RB_{1:n}SE_{1:(m-1)}$.

($\Leftarrow$) It is shown that, if $D_{1:(n+m-1)}$ passes $RB_{1:n}SE_{1:(m-1)}$, then $D_{1:(n+m-1)}$ fits $F_{1:(n+m-1)}$, and $RB_{1:n}SE_{1:(m-1)}$ realizes the permutation represented by $D_{m:(n+m-1)}$.

In order for a matrix of order $N \times (n+m-1)$ to fit $F_{1:(n+m-1)}$, the two necessary and sufficient conditions are (1) the matrix is balanced, and (2) the first $n-1$ columns of the matrix fit $F_{1:(n-1)}$. As for $D_{1:(n+m-1)}$, it is a given *balanced* matrix. Because $D_{1:(n+m-1)}$ passes $RB_{1:n}SE_{1:(m-1)}$ by hypothesis, the submatrix $D_{1:n}$ of $D_{1:(n+m-1)}$ passes $RB_{1:n}$. So, by Theorem 3.5, the submatrix $D_{1:n}$ fits $F_{1:n}$, which implies that $D_{1:(n-1)}$ fits $F_{1:(n-1)}$. So $D_{1:(n+m-1)}$ satisfies the above two conditions, and hence it fits $F_{1:(n+m-1)}$.

The first part ($\Rightarrow$) of the proof has shown that the permutation represented by $D_{m:(n+m-1)}$ is realized by $RB_{1:n}SE_{1:(m-1)}$ if $D_{1:(n+m-1)}$ fits $F_{1:(n+m-1)}$. Because it is shown above that $D_{1:(n+m-1)}$ fits $F_{1:(n+m-1)}$, $RB_{1:n}SE_{1:(m-1)}$ realizes the permutation represented by $D_{m:(n+m-1)}$. □

Proof of Lemma 5.1. We begin by showing first that, if $[R_{1:n}\ D_{1:n}]$ is balanced, then $D_{1:n}$ fits $F_{1:n}$. Let $1 \le j \le n$. Partition the balanced matrix $[R_{(j+1):n}\ D_{1:j}]$ into 2^{n-j} submatrices $S^h_{2^j \times n} = [R^h_{(j+1):n}\ D^h_{1:j}]$, $0 \le h \le 2^{n-j}-1$, such that each of $S^h_{2^j \times n}$, $R^h_{(j+1):n}$, and $D^h_{1:j}$ contains 2^j rows whose labels belong to the rows of the same block under column f_j of $F_{1:n}$, where the row labels of $S^h_{2^j \times n}$, $R^h_{(j+1):n}$, and $D^h_{1:j}$ consist of the numbers $h \times 2^j$ to $[(h+1) \times 2^j] - 1$ inclusive. (As an example, for $n = 3$ and $j = 1$, the row labels of submatrices $R^0_{2:3}$, $R^1_{2:3}$, $R^2_{2:3}$, and $R^3_{2:3}$, form the sets $\{0,1\}$, $\{2,3\}$, $\{4,5\}$, and $\{6,7\}$, respectively. (As an example, for $n = 3$ and $j = 2$, the row labels of submatrices $R^0_{3:3}$ and $R^1_{3:3}$ form the sets $\{0,1,2,3\}$ and $\{4,5,6,7\}$, respectively.) It follows from the definition of the reverse permutation matrix $R_{1:n}$ (Definition 2.1) that the row contents of $R^h_{(j+1):n}$ are identical. Moreover, since $[R_{1:n}\ D_{1:n}]$ is balanced by hypothesis, $[R_{(j+1):n}\ D_{1:j}]$ must also be balanced because of the definition of a balanced matrix. Therefore, $D^h_{1:j}$ must be a balanced matrix of order $2^j \times j$ (because $R^h_{(j+1):n}$ are identical as mentioned above). Since this is true for all values of j and

h, it follows from the definitions of $F_{1:n}$ and "fit" that $D_{1:n}$ fits $F_{1:n}$.

By Theorem 3.5, a matrix $D_{1:n}$ fits $F_{1:n}$ if and only if the matrix $D_{1:n}$ is realized by $RB_{1:n}$. Also, it is shown above that, if $[R_{1:n}\ D_{1:n}]$ is balanced, then $D_{1:n}$ fits $F_{1:n}$. Therefore, if $[R_{1:n}\ D_{1:n}]$ is balanced, then $D_{1:n}$ is realized by $RB_{1:n}$. On the other hand, it follows from Lemma 3.6 that if $[I_{1:n}\ D_{1:n}]$ is balanced, then $D_{1:n}$ is realized by $SE_{1:n}$.

Relabeling the ith input of $RB_{1:n}$ (respectively, $SE_{1:n}$) by the value of $R_{1:n}(i)$ is equivalent to moving the row of $R_{1:n}$ (respectively, $I_{1:n}$) with a label equal to $R_{1:n}(i)$ to the location of the ith row. Also, note that if $R_{1:n}(i) = j$, then $R_{1:n}(j) = i$ for $0 \le i, j \le N-1$. Therefore, the rearrangement of the rows of $R_{1:n}$ and $I_{1:n}$ due to relabeling results in the matrices $I_{1:n}$ and $R_{1:n}$, respectively. This implies that, after the relabeling, the matrix $[I_{1:n}\ D_{1:n}]$ (respectively, $[R_{1:n}\ D_{1:n}]$) must be balanced in order for a given matrix $D_{1:n}$ to be realized by $RB_{1:n}$ (respectively, $SE_{1:n}$). Thus relabeling the inputs of $RB_{1:n}$ and $SE_{1:n}$ in the way described above enables them to exactly simulate each other. It follows that the lemma holds. □

Acknowledgments. I am very grateful to Jose A. B. Fortes for his very valuable comments and the very generous use of his time in painstakingly checking on the correctness of the proofs in the preliminary versions of this paper. Without the support and guidance of Jose A. B. Fortes while he was my Ph.D. thesis advisor at Purdue University, I would not have been able to come up with some proofs presented in this paper. I also thank Frank K. Hwang of Chiaotung University for his constructive comments that helped improve the technical presentation of the paper. Finally, I am grateful to the anonymous referees for their very constructive comments that greatly improved the paper.

REFERENCES

[1] D. H. Lawrie, *Access and alignment of data in an array processor*, IEEE Trans. Comput., 24 (1975), pp. 1145–1155.

[2] A. Varma and C. S. Raghavendra, *Rearrangeability of multistage shuffle/exchange networks*, in Proceedings of the 14th Annual Symposium on Computer Architecture, Pittsburgh, PA, 1987, pp. 154–162.

[3] C. S. Raghavendra and A. Varma, *Rearrangeability of the five stage shuffle-exchange network for $N = 8$*, IEEE Trans. Commun., 35 (1987), pp. 808–812.

[4] N. Linial and M. Tarsi, *Interpolation between bases and the shuffle-exchange network*, European J. Combin., 10 (1989), pp. 29–39.

[5] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, American Elsevier, New York, 1976.

[6] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss, *The IBM Research Parallel Processor Prototype* (RP3): *Introduction and architecture*, in Proceedings of the IEEE International Conference on Parallel Processing, Chicago, IL, 1985, pp. 764–771.

[7] H. S. Stone, *Parallel processing with the perfect shuffle*, IEEE Trans. Comput., 20 (1971), pp. 153–161.

[8] D. S. Parker, *Notes on shuffle/exchange-type switching networks*, IEEE Trans. Comput., 29 (1980), pp. 213–222.

[9] V. E. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, 1965.

[10] A. Waksman, *A permutation network*, J. ACM, 15 (1969), pp. 159–163.

[11] C. P. Kruskal and M. Snir, *A unified theory of interconnection network structure*, Theoret. Comput. Sci., 48 (1986), pp. 75–94.

[12] H. J. Siegel, *The universality of various types of SIMD machine interconnection networks*, in Proceedings of the 4th Annual Symposium on Computer Architecture, Silver Spring, MD, 1977, pp. 70–79.

[13] C. Wu and T. Y. Feng, *On a class of multistage interconnection networks*, IEEE Trans. Comput., 29 (1980), pp. 696–702.
[14] T. Lang, *Interconnections between processors and memory modules using the shuffle-exchange network*, IEEE Trans. Comput., 25 (1976), pp. 496–503.
[15] T. Etzion and A. Lempel, *An efficient algorithm for generating linear transformations in a shuffle-exchange network*, SIAM J. Comput., 15 (1986), pp. 216–221.
[16] C. S. Babu and C. S. Raghavendra, *On the invariants of shuffle/exchange networks*, in Proceedings of the 1st IEEE Symposium on Parallel and Distributed Processing, IEEE Computer Society, Los Alamitos, CA, 1989, pp. 249–256.
[17] A. Y. Oruc and M. Y. Oruc, *Equivalence relations among interconnection networks*, J. Parallel Distributed Comput., 2 (1985), pp. 30–49.
[18] C. Wu and T.-Y. Feng, *The universality of the shuffle-exchange network*, IEEE Trans. Comput., 30 (1981), pp. 324–331.
[19] S. T. Huang and S. K. Tripathi, *Finite state model and compatibility theory: New analysis tools for permutation networks*, IEEE Trans. Comput., 35 (1986), pp. 591–601.
[20] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, *The NYU ultracomputer—designing an MIMD shared memory parallel computer*, IEEE Trans. Comput., 32 (1983), pp. 175–189.
[21] C. S. Raghavendra, *On the rearrangeability conjecture of* $(2\log_2 N - 1)$*-stage shuffle/exchange network*, position paper, Computer Architecture Technical Committee Newsletter, Winter 1994–1995, pp. 10–12.
[22] H. Çam and J. A. B. Fortes, *Frames: A simple characterization of permutations realized by frequently used networks*, IEEE Trans. Comput., 44 (1995), pp. 695–697.
[23] H. Çam and J. A. B. Fortes, *Frames: A Simple Characterization of Permutations Realized by Frequently Used Networks*, Technical report TR-EE 92-32, Purdue University, West Lafayette, IN, 1992.
[24] T.-Y. Feng and S.-W. Seo, *A new routing algorithm for a class of rearrangeable networks*, IEEE Trans. Comput., 43 (1994), pp. 1270–1280.
[25] M. K. Kim, H. Yoon, and S. R. Maeng, *On the correctness of inside-out routing algorithm*, IEEE Trans. Comput., 46 (1997), pp. 820–823.
[26] V. E. Beneš, *Proving the rearrangeability of connecting networks by group calculations*, Bell System Tech. J., 45 (1975), pp. 421–434.
[27] F. K. Hwang, *A mathematical abstraction of the rearrangeability conjecture for shuffle-exchange networks*, Oper. Res. Lett., 8 (1989), pp. 85–87.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 586–615

IDENTITY-BASED ENCRYPTION FROM THE WEIL PAIRING*

DAN BONEH† AND MATTHEW FRANKLIN‡

Abstract. We propose a fully functional identity-based encryption (IBE) scheme. The scheme has chosen ciphertext security in the random oracle model assuming a variant of the computational Diffie–Hellman problem. Our system is based on bilinear maps between groups. The Weil pairing on elliptic curves is an example of such a map. We give precise definitions for secure IBE schemes and give several applications for such systems.

Key words. identity-based encryption, bilinear maps, Weil pairing, Tate pairing, elliptic curve cryptography, escrow ElGamal

AMS subject classifications. 94A60, 68P25, 14G50, 11T71

PII. S0097539701398521

1. Introduction.

In 1984, Shamir [41] asked for a public key encryption scheme in which the public key can be an arbitrary string. In such a scheme, there are four algorithms: (1) Setup generates global system parameters and a master-key; (2) Extract uses the master-key to generate the private key corresponding to an arbitrary public key string $\mathsf{ID} \in \{0,1\}^*$; (3) Encrypt encrypts messages using the public key ID; and (4) Decrypt decrypts messages using the corresponding private key.

Shamir's original motivation for suggesting identity-based encryption (IBE) was to simplify certificate management in e-mail systems. When Alice sends mail to Bob at `bob@company.com`, she simply encrypts her message using the public key string "`bob@company.com`." There is no need for Alice to obtain Bob's public key certificate. When Bob receives the encrypted mail, he contacts a third party, which we call the private key generator (PKG). Bob authenticates himself to the PKG in the same way he would authenticate himself to a certificate authority (CA) and obtains his private key from the PKG. Bob can then read his e-mail. Note that, unlike in the existing secure e-mail infrastructure, Alice can send encrypted mail to Bob even if Bob has not yet set up his public key certificate. Also note that key escrow is inherent in identity-based e-mail systems: the PKG knows Bob's private key. We discuss key revocation, as well as several new applications for IBE schemes, in the next section.

Since the problem was posed in 1984, there have been several proposals for IBE schemes [11, 45, 44, 31, 25] (see also [33, p. 561]). However, none of these are fully satisfactory. Some solutions require that users not collude. Other solutions require the PKG to spend a long time for each private key generation request. Some solutions require tamper resistant hardware. It is fair to say that, until the results in [5], constructing a usable IBE system was an open problem. Interestingly, the related notions of identity-based signature and authentication schemes, also introduced by Shamir [41], do have satisfactory solutions [15, 14].

*Received by the editors October 14, 2001; accepted for publication (in revised form) October 4, 2002; published electronically March 5, 2003. This is the full version of an earlier paper published in Crypto 2001, Lecture Notes in Comput. Sci. 2139, Springer-Verlag, Berlin, 2001, pp. 213–229.
http://www.siam.org/journals/sicomp/32-3/39852.html

†Department of Computer Science, Stanford University, Stanford, CA 94305 (dabo@cs.stanford.edu). This author's research was supported by DARPA, an NSF Career Award, and the Packard Foundation.

‡Department of Computer Science, University of California at Davis, Davis, CA 95616 (franklin@cs.ucdavis.edu). This author's research was supported by an NSF Career Award and the Packard Foundation.

In this paper, we propose a fully functional IBE scheme. The performance of our system is comparable to the performance of ElGamal encryption in $\mathbb{F}_p^*$. The security of our system is based on a natural analogue of the computational Diffie–Hellman assumption. Based on this assumption, we show that the new system has chosen ciphertext security in the random oracle model. Using standard techniques from threshold cryptography [20, 22], the PKG in our scheme can be distributed so that the master-key is never available in a single location. Unlike common threshold systems, we show that robustness for our distributed PKG is free.

Our IBE system can be built from any bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ between two groups $\mathbb{G}_1, \mathbb{G}_2$ as long as a variant of the computational Diffie–Hellman problem (CDH) in $\mathbb{G}_1$ is hard. We use the Weil pairing on elliptic curves as an example of such a map. Until recently, the Weil pairing has mostly been used for attacking elliptic curve systems [32, 17]. Joux [26] recently showed that the Weil pairing can be used for "good" by using it for a protocol for three party one round Diffie–Hellman key exchange. Sakai, Ohgishi, and Kasahara [40] used the pairing for key exchange, and Verheul [46] used it to construct an ElGamal encryption scheme where each public key has two corresponding private keys. In addition to our IBE scheme, we show how to construct an ElGamal encryption scheme with "built-in" key escrow, i.e., where one global escrow key can decrypt ciphertexts encrypted under any public key.

To argue about the security of our IBE system, we define chosen ciphertext security for IBE. Our model gives the adversary more power than the standard model for chosen ciphertext security [37, 2]. First, we allow the attacker to attack an arbitrary public key ID of her choice. Second, while mounting a chosen ciphertext attack on ID, we allow the attacker to obtain from the PKG the private key for any public key of her choice, other than the private key for ID. This models an attacker who obtains a number of private keys corresponding to some identities of her choice and then tries to attack some other public key ID of her choice. Even with the help of such queries, the attacker should have negligible advantage in defeating the semantic security of the system.

The rest of the paper is organized as follows. Several applications of IBE are discussed in section 1.1. We then give precise definitions and security models in section 2. We describe bilinear maps with certain properties in section 3. Our IBE scheme is presented in section 4 using general bilinear maps. Then a concrete identity-based system from the Weil pairing is given in section 5. Some extensions and variations (efficiency improvements, distribution of the master-key) are considered in section 6. Our construction for ElGamal encryption with a global escrow key is described in section 7. Section 8 gives conclusions and some open problems. The appendices contain a more detailed discussion of the Weil pairing.

1.1. Applications for IBE. The original motivation for IBE is to help the deployment of a public key infrastructure. In this section, we show several other unrelated applications.

1.1.1. Revocation of public keys. Public key certificates contain a preset expiration date. In an IBE system, key expiration can be done by having Alice encrypt e-mail sent to Bob using the public key "`bob@company.com` $\|$ `current-year`." In doing so, Bob can use his private key during the current year only. Once a year, Bob needs to obtain a new private key from the PKG. Hence we get the effect of annual private key expiration. Note that unlike the existing public key infrastructure (PKI), Alice does not need to obtain a new certificate from Bob every time Bob refreshes his private key.

One could potentially make this approach more granular by encrypting e-mail for Bob using "bob@company.com ‖ current-date." This forces Bob to obtain a new private key every day. This might be possible in a corporate PKI, where the PKG is maintained by the corporation. With this approach, key revocation is very simple: when Bob leaves the company and his key needs to be revoked, the corporate PKG is instructed to stop issuing private keys for Bob's e-mail address. As a result, Bob can no longer read his email. The interesting property is that Alice does not need to communicate with any third party certificate directory to obtain Bob's daily public key. Hence IBE is a very efficient mechanism for implementing ephemeral public keys. Also note that this approach enables Alice to send messages into the future: Bob will only be able to decrypt the e-mail on the date specified by Alice (see [38, 12] for methods of sending messages into the future using a stronger security model).

Managing user credentials. A simple extension to the discussion above enables us to manage user credentials using the IBE system. Suppose Alice encrypts mail to Bob using the public key: "bob@company.com ‖ current-year ‖ clearance=secret." Then Bob will only be able to read the e-mail if on the specified date he has secret clearance. Consequently, it is easy to grant and revoke user credentials using the PKG.

1.1.2. Delegation of decryption keys. Another application for IBE systems is delegation of decryption capabilities. We give two example applications. In both applications, the user Bob plays the role of the PKG. Bob runs the Setup algorithm to generate his own IBE system parameters params and his own master-key. Here we view params as Bob's public key. Bob obtains a certificate from a CA for his public key params. When Alice wishes to send mail to Bob, she first obtains Bob's public key params from Bob's public key certificate. Note that Bob is the only one who knows his master-key, and hence there is no key escrow with this setup.

1. *Delegation to a laptop.* Suppose Alice encrypts mail to Bob using the current date as the IBE encryption key. (She uses Bob's params as the IBE system parameters.) Since Bob has the master-key, he can extract the private key corresponding to this IBE encryption key and then decrypt the message. Now, suppose Bob goes on a trip for seven days. Normally, Bob would put his private key on his laptop. If the laptop is stolen, the private key is compromised. When using the IBE system, Bob could simply install on his laptop the seven private keys corresponding to the seven days of the trip. If the laptop is stolen, only the private keys for those seven days are compromised. The master-key is unharmed. This is analogous to the delegation scenario for *signature schemes* considered by Goldreich, Pfitzmann, and Rivest [23].

2. *Delegation of duties.* Suppose Alice encrypts mail to Bob using the subject line as the IBE encryption key. Bob can decrypt mail using his master-key. Now, suppose Bob has several assistants each responsible for a different task (e.g., one is "purchasing," another is "human-resources," etc.). Bob gives one private key to each of his assistants corresponding to the assistant's responsibility. Each assistant can then decrypt messages whose subject line falls within its responsibilities, but it cannot decrypt messages intended for other assistants. Note that Alice obtains only a single public key from Bob (params), and she uses that public key to send mail with any subject line of her choice. The mail can be read only by the assistant responsible for that subject.

More generally, IBE can simplify security systems that manage a large number of public keys. Rather than storing a big database of public keys, the system can either derive these public keys from usernames or simply use the integers $1, \ldots, n$ as

distinct public keys.

2. Definitions. IBE. An IBE scheme $\mathcal{E}$ is specified by four randomized algorithms: Setup, Extract, Encrypt, and Decrypt.

Setup takes a security parameter k and returns params (system parameters) and master-key. The system parameters include a description of a finite message space $\mathcal{M}$ and a description of a finite ciphertext space $\mathcal{C}$. Intuitively, the system parameters will be publicly known, while the master-key will be known only to the "private key generator (PKG)."

Extract takes as input params, master-key, and an arbitrary $\mathsf{ID} \in \{0,1\}^*$ and returns a private key d. Here ID is an arbitrary string that will be used as a public key, and d is the corresponding private decryption key. The Extract algorithm extracts a private key from the given public key.

Encrypt takes as input params, ID, and $M \in \mathcal{M}$. It returns a ciphertext $C \in \mathcal{C}$.

Decrypt takes as input params, $C \in \mathcal{C}$, and a private key d. It returns $M \in \mathcal{M}$.

These algorithms must satisfy the standard consistency constraint; namely, when d is the private key generated by algorithm Extract when it is given ID as the public key, then

$$\forall M \in \mathcal{M} : \mathsf{Decrypt}(\mathsf{params}, C, d) = M, \quad \text{where} \quad C = \mathsf{Encrypt}(\mathsf{params}, \mathsf{ID}, M).$$

Chosen ciphertext security. Chosen ciphertext security (IND-CCA) is the standard acceptable notion of security for a public key encryption scheme [37, 2, 13]. Hence it is natural to require that an IBE scheme also satisfy this strong notion of security. However, the definition of chosen ciphertext security must be strengthened a bit. The reason is that when an adversary attacks a public key ID in an identity-based system, the adversary might already possess the private keys of users $\mathsf{ID}_1, \ldots, \mathsf{ID}_n$ of her choice. The system should remain secure under such an attack. Hence the definition of chosen ciphertext security must allow the adversary to obtain the private key associated with any identity ID_i of her choice (other than the public key ID being attacked). We refer to such queries as private key extraction queries. Another difference is that the adversary is challenged on a public key ID of her choice (as opposed to a random public key).

We say that an IBE scheme $\mathcal{E}$ is semantically secure against an adaptive chosen ciphertext attack (IND-ID-CCA) if no polynomially bounded adversary $\mathcal{A}$ has a nonnegligible advantage against the challenger in the following IND-ID-CCA game.

Setup. The challenger takes a security parameter k and runs the Setup algorithm. It gives the adversary the resulting system parameters params. It keeps the master-key to itself.

Phase 1. The adversary issues queries $q_1, \ldots, q_m$, where query q_i is one of the following:

- Extraction query $\langle \mathsf{ID}_i \rangle$. The challenger responds by running algorithm Extract to generate the private key d_i corresponding to the public key $\langle \mathsf{ID}_i \rangle$. It sends d_i to the adversary.
- Decryption query $\langle \mathsf{ID}_i, C_i \rangle$. The challenger responds by running algorithm Extract to generate the private key d_i corresponding to ID_i. It then runs algorithm Decrypt to decrypt the ciphertext C_i using the private key d_i. It sends the resulting plaintext to the adversary.

These queries may be asked adaptively; that is, each query q_i may depend on the replies to $q_1, \ldots, q_{i-1}$.

Challenge. Once the adversary decides that Phase 1 is over, it outputs two equal length plaintexts $M_0, M_1 \in \mathcal{M}$ and an identity ID on which it wishes to be challenged. The only constraint is that ID did not appear in any private key extraction query in Phase 1.

The challenger picks a random bit $b \in \{0,1\}$ and sets the challenge ciphertext to $C = \mathsf{Encrypt}(\mathsf{params}, \mathsf{ID}, M_b)$. It sends C as the challenge to the adversary.

Phase 2. The adversary issues more queries $q_{m+1}, \ldots, q_n$, where q_i is one of the following:

- Extraction query $\langle \mathsf{ID}_i \rangle$, where $\mathsf{ID}_i \neq \mathsf{ID}$. The challenger responds as in Phase 1.
- Decryption query $\langle \mathsf{ID}_i, C_i \rangle \neq \langle \mathsf{ID}, C \rangle$. The challenger responds as in Phase 1.

These queries may be asked adaptively as in Phase 1.

Guess. Finally, the adversary outputs a guess $b' \in \{0,1\}$. The adversary wins the game if $b = b'$.

We refer to such an adversary $\mathcal{A}$ as an IND-ID-CCA adversary. We define adversary $\mathcal{A}$'s advantage in attacking the scheme $\mathcal{E}$ as the following function of the security parameter k (recall that k is given as input to the challenger):

$$\mathrm{Adv}_{\mathcal{E},\mathcal{A}}(k) = \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

The probability is over the random bits used by the challenger and the adversary.

Using the IND-ID-CCA game, we can define chosen ciphertext security for IBE schemes. As usual, we say that a function $g : \mathbb{R} \to \mathbb{R}$ is *negligible* if $g(k)$ is smaller than $1/f(k)$ for any polynomial f.

DEFINITION 2.1. *We say that the IBE system $\mathcal{E}$ is semantically secure against an adaptive chosen ciphertext attack if for any polynomial time* IND-ID-CCA *adversary $\mathcal{A}$ the function $Adv_{\mathcal{E},\mathcal{A}}(k)$ is negligible. As shorthand, we say that $\mathcal{E}$ is* IND-ID-CCA *secure.*

Note that the standard definition of chosen ciphertext security (IND-CCA) [37, 2] is the same as above except that there are no private key extraction queries and the adversary is challenged on a random public key (rather than a public key of her choice). Private key extraction queries are related to the definition of chosen ciphertext security in the multiuser settings [7]. After all, our definition involves multiple public keys belonging to multiple users. In [7], the authors show that that multiuser IND-CCA is reducible to single user IND-CCA using a standard hybrid argument. This does not hold in the identity-based settings, IND-ID-CCA, since the adversary gets to choose which public keys to corrupt during the attack. To emphasize the importance of private key extraction queries, we note that our IBE system can be easily modified (by removing one of the hash functions) into a system which has chosen ciphertext security when private extraction queries are disallowed. However, the scheme is completely insecure when extraction queries are allowed.

Semantically secure IBE. The proof of security for our IBE system makes use of a weaker notion of security known as semantic security (also known as semantic security against a chosen plaintext attack) [24, 2]. Semantic security is similar to chosen ciphertext security (IND-ID-CCA) except that the adversary is more limited; it cannot issue decryption queries. For a standard public key system (not an identity-based system), semantic security is defined using the following game: (1) the adversary is given a random public key generated by the challenger; (2) the adversary outputs

two equal-length messages M_0 and M_1 and receives the encryption of M_b from the challenger, where b is chosen at random in $\{0,1\}$; (3) the adversary outputs b' and wins the game if $b = b'$. The public key system is said to be semantically secure if no polynomial time adversary can win the game with a nonnegligible advantage. As shorthand, we say that a semantically secure public key system is IND-CPA secure. Semantic security captures our intuition that, given a ciphertext, the adversary learns nothing about the corresponding plaintext.

To define semantic security for identity-based systems (denoted IND-ID-CPA), we strengthen the standard definition by allowing the adversary to issue chosen private key extraction queries. Similarly, the adversary is challenged on a public key ID of her choice. We define semantic security for IBE schemes using an IND-ID-CPA game. The game is identical to the IND-ID-CCA game defined above except that the adversary cannot make any decryption queries. The adversary can only make private key extraction queries. We say that an IBE scheme $\mathcal{E}$ is semantically secure (IND-ID-CPA) if no polynomially bounded adversary $\mathcal{A}$ has a nonnegligible advantage against the challenger in the following IND-ID-CPA game.

Setup. The challenger takes a security parameter k and runs the Setup algorithm. It gives the adversary the resulting system parameters params. It keeps the master-key to itself.

Phase 1. The adversary issues private key extraction queries $\mathsf{ID}_1, \ldots, \mathsf{ID}_m$. The challenger responds by running algorithm Extract to generate the private key d_i corresponding to the public key ID_i. It sends d_i to the adversary. These queries may be asked adaptively.

Challenge. Once the adversary decides that Phase 1 is over, it outputs two equal-length plaintexts $M_0, M_1 \in \mathcal{M}$ and a public key ID on which it wishes to be challenged. The only constraint is that ID did not appear in any private key extraction query in Phase 1. The challenger picks a random bit $b \in \{0,1\}$ and sets $C = \mathsf{Encrypt}(\mathsf{params}, \mathsf{ID}, M_b)$. It sends C as the challenge to the adversary.

Phase 2. The adversary issues more extraction queries $\mathsf{ID}_{m+1}, \ldots, \mathsf{ID}_n$. The only constraint is that $\mathsf{ID}_i \neq \mathsf{ID}$. The challenger responds as in Phase 1.

Guess. Finally, the adversary outputs a guess $b' \in \{0,1\}$. The adversary wins the game if $b = b'$.

We refer to such an adversary $\mathcal{A}$ as an IND-ID-CPA adversary. As in the above, the advantage of an IND-ID-CPA adversary $\mathcal{A}$ against the scheme $\mathcal{E}$ is the following function of the security parameter k: $\mathrm{Adv}_{\mathcal{E},\mathcal{A}}(k) = \left|\Pr[b = b'] - \frac{1}{2}\right|$.

The probability is over the random bits used by the challenger and the adversary.

DEFINITION 2.2. *We say that the IBE system $\mathcal{E}$ is semantically secure if for any polynomial time* IND-ID-CPA *adversary $\mathcal{A}$ the function $Adv_{\mathcal{E},\mathcal{A}}(k)$ is negligible. As shorthand, we say that $\mathcal{E}$ is* IND-ID-CPA *secure.*

One-way IBE. One can define an even weaker notion of security called one-way encryption (OWE) [16]. Roughly speaking, a public key encryption scheme is a one-way encryption if, given the encryption of a random plaintext, the adversary cannot produce the plaintext in its entirety. One-way encryption is a weak notion of security since there is nothing preventing the adversary from, say, learning half the bits of the plaintext. Hence one-way encryption schemes do not generally provide secure encryption. In the random oracle model, one-way encryption schemes can be used for encrypting session-keys. (The session-key is taken to be the hash of the plaintext.) We note that one can extend the notion of one-way encryption to identity-based systems

by adding private key extraction queries to the definition. We do not give the full definition here since in this paper we use semantic security as the weakest notion of security. See [5] for the full definition of identity-based one-way encryption and its use as part of an alternative proof strategy for our main result.

Random oracle model. To analyze the security of certain natural cryptographic constructions, Bellare and Rogaway introduced an idealized security model called the random oracle model [3]. Roughly speaking, a random oracle is a function $H : X \to Y$ chosen uniformly at random from the set of all functions $\{h : X \to Y\}$. (We assume Y is a finite set.) An algorithm can query the random oracle at any point $x \in X$ and receive the value $H(x)$ in response. Random oracles are used to model cryptographic hash functions such as SHA-1. Note that security in the random oracle model does not imply security in the real world. Nevertheless, the random oracle model is a useful tool for validating natural cryptographic constructions. Security proofs in this model prove security against attackers that are confined to the random oracle world.

Notation. From here on we use $\mathbb{Z}_q$ to denote the group $\{0, \ldots, q-1\}$ under addition modulo q. For a group $\mathbb{G}$ of prime order, we use $\mathbb{G}^*$ to denote the set $\mathbb{G}^* = \mathbb{G} \setminus \{O\}$, where O is the identity element in the group $\mathbb{G}$. We use $\mathbb{Z}^+$ to denote the set of positive integers.

3. Bilinear maps and the bilinear Diffie–Hellman assumption. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of order q for some large prime q. Our IBE system makes use of a *bilinear* map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ between these two groups. The map must satisfy the following properties:

1. *Bilinear.* We say that a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is *bilinear* if $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$ for all $P, Q \in \mathbb{G}_1$ and all $a, b \in \mathbb{Z}$.
2. *Nondegenerate.* The map does not send all pairs in $\mathbb{G}_1 \times \mathbb{G}_1$ to the identity in $\mathbb{G}_2$. Observe that since $\mathbb{G}_1, \mathbb{G}_2$ are groups of prime order, this implies that if P is a generator of $\mathbb{G}_1$, then $\hat{e}(P, P)$ is a generator of $\mathbb{G}_2$.
3. *Computable.* There is an efficient algorithm to compute $\hat{e}(P, Q)$ for any $P, Q \in \mathbb{G}_1$.

A bilinear map satisfying the three properties above is said to be an *admissible* bilinear map. In section 5, we give a concrete example of groups $\mathbb{G}_1, \mathbb{G}_2$ and an admissible bilinear map between them. The group $\mathbb{G}_1$ is a subgroup of the additive group of points of an elliptic curve $E/\mathbb{F}_p$. The group $\mathbb{G}_2$ is a subgroup of the multiplicative group of a finite field $\mathbb{F}^*_{p^2}$. Therefore, throughout the paper, we view $\mathbb{G}_1$ as an additive group and $\mathbb{G}_2$ as a multiplicative group. As we will see in section 5.1, the Weil pairing can be used to construct an admissible bilinear map between these two groups.

The existence of the bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ as above has two direct implications to these groups.

1. *The MOV reduction.* Menezes, Okamoto, and Vanstone [32] show that the discrete log problem in $\mathbb{G}_1$ is no harder than the discrete log problem in $\mathbb{G}_2$. To see this, let $P, Q \in \mathbb{G}_1$ be an instance of the discrete log problem in $\mathbb{G}_1$ where both P, Q have order q. We wish to find an $\alpha \in \mathbb{Z}_q$ such that $Q = \alpha P$. Let $g = \hat{e}(P, P)$ and $h = \hat{e}(Q, P)$. Then, by bilinearity of $\hat{e}$, we know that $h = g^\alpha$. By nondegeneracy of $\hat{e}$, both g, h have order q in $\mathbb{G}_2$. Hence we reduced the discrete log problem in $\mathbb{G}_1$ to a discrete log problem in $\mathbb{G}_2$. It follows that, for discrete log to be hard in $\mathbb{G}_1$, we must choose our security parameter so that discrete log is hard in $\mathbb{G}_2$ (see section 5).

2. *Decision Diffie–Hellman is easy.* The decision Diffie–Hellman problem (DDH) [4] in $\mathbb{G}_1$ is to distinguish between the distributions $\langle P, aP, bP, abP\rangle$ and $\langle P, aP, bP, cP\rangle$, where a, b, c are random in $\mathbb{Z}^*_q$ and P is random in $\mathbb{G}^*_1$. Joux and Nguyen [28] point

out that DDH in $\mathbb{G}_1$ is easy. To see this, observe that, given $P, aP, bP, cP \in \mathbb{G}_1^*$, we have

$$c = ab \bmod q \quad \Longleftrightarrow \quad \hat{e}(P, cP) = \hat{e}(aP, bP).$$

The computational Diffie–Hellman problem (CDH) in $\mathbb{G}_1$ can still be hard. (CDH in G_1 is to find abP given random $\langle P, aP, bP\rangle$.) Joux and Nguyen [28] give examples of mappings $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, where CDH in $\mathbb{G}_1$ is believed to be hard even though DDH in $\mathbb{G}_1$ is easy.

3.1. The bilinear Diffie–Hellman assumption (BDH). Since the DDH in $\mathbb{G}_1$ is easy, we cannot use DDH to build cryptosystems in the group $\mathbb{G}_1$. Instead, the security of our IBE system is based on a variant of the CDH assumption called the bilinear Diffie–Hellman assumption (BDH).

BDH problem. Let $\mathbb{G}_1, \mathbb{G}_2$ be two groups of prime order q. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ be an admissible bilinear map, and let P be a generator of $\mathbb{G}_1$. The BDH problem in $\langle \mathbb{G}_1, \mathbb{G}_2, \hat{e}\rangle$ is as follows: Given $\langle P, aP, bP, cP\rangle$ for some $a, b, c \in \mathbb{Z}_q^*$, compute $W = \hat{e}(P, P)^{abc} \in \mathbb{G}_2$. An algorithm $\mathcal{A}$ has advantage ϵ in solving BDH in $\langle \mathbb{G}_1, \mathbb{G}_2, \hat{e}\rangle$ if

$$\Pr\left[\mathcal{A}(P, aP, bP, cP) = \hat{e}(P, P)^{abc}\right] \geq \epsilon,$$

where the probability is over the random choice of a, b, c in $\mathbb{Z}_q^*$, the random choice of $P \in \mathbb{G}_1^*$, and the random bits of $\mathcal{A}$.

BDH parameter generator. We say that a randomized algorithm $\mathcal{G}$ is a *BDH parameter generator* if (1) $\mathcal{G}$ takes a security parameter $k \in \mathbb{Z}^+$, (2) $\mathcal{G}$ runs in polynomial time in k, and (3) $\mathcal{G}$ outputs a prime number q, the description of two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, and the description of an admissible bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. We denote the output of $\mathcal{G}$ by $\mathcal{G}(1^k) = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}\rangle$. The security parameter k is used to determine the size of q; for example, one could take q to be a random k-bit prime. For $i = 1, 2$, we assume that the description of the group $\mathbb{G}_i$ contains polynomial time (in k) algorithms for computing the group action in $\mathbb{G}_i$ and contains a generator of $\mathbb{G}_i$. The generator of $\mathbb{G}_i$ enables us to generate uniformly random elements in $\mathbb{G}_i$. Similarly, we assume that the description of $\hat{e}$ contains a polynomial time algorithm for computing $\hat{e}$. We give an example of a BDH parameter generator in section 5.1.

BDH assumption. Let $\mathcal{G}$ be a BDH parameter generator. We say that an algorithm $\mathcal{A}$ has advantage $\epsilon(k)$ in solving the BDH problem for $\mathcal{G}$ if, for sufficiently large k,

$$\mathrm{Adv}_{\mathcal{G},\mathcal{A}}(k) = \Pr\left[\mathcal{A}\begin{pmatrix} q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, \\ P, aP, bP, cP \end{pmatrix} = \hat{e}(P, P)^{abc} \;\middle|\; \begin{array}{l} \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}\rangle \leftarrow \mathcal{G}(1^k), \\ P \leftarrow \mathbb{G}_1^*,\ a, b, c \leftarrow \mathbb{Z}_q^* \end{array}\right] \geq \epsilon(k).$$

We say that $\mathcal{G}$ satisfies the BDH assumption if for any randomized polynomial time (in k) algorithm $\mathcal{A}$ and for any polynomial $f \in \mathbb{Z}[x]$ we have that $\mathrm{Adv}_{\mathcal{G},\mathcal{A}}(k) < 1/f(k)$ for sufficiently large k. When $\mathcal{G}$ satisfies the BDH assumption, we say that BDH is hard in groups generated by $\mathcal{G}$.

In section 5.1, we give some examples of BDH parameter generators that are believed to satisfy the BDH assumption. We note that Joux [26] (implicitly) used the BDH assumption to construct a one-round three party Diffie–Hellman protocol. The BDH assumption is also needed for constructions in [46, 40].

Hardness of BDH. It is interesting to study the relationship of the BDH problem to other hard problems used in cryptography. Currently, all we can say is that the

BDH problem in $\langle \mathbb{G}_1, \mathbb{G}_2, \hat{e} \rangle$ is no harder than the CDH problem in $\mathbb{G}_1$ or $\mathbb{G}_2$. In other words, an algorithm for CDH in $\mathbb{G}_1$ or $\mathbb{G}_2$ is sufficient for solving BDH in $\langle \mathbb{G}_1, \mathbb{G}_2, \hat{e} \rangle$. The converse is currently an open problem: is an algorithm for BDH sufficient for solving CDH in $\mathbb{G}_1$ or in $\mathbb{G}_2$? We refer to a recent survey by Joux [27] for a more detailed analysis of the relationship between BDH and other standard problems.

We note that in all our examples (in section 5.1) the isomorphisms from $\mathbb{G}_1$ to $\mathbb{G}_2$ induced by the bilinear map are believed to be one-way functions. More specifically, for a point $Q \in \mathbb{G}_1^*$ define the isomorphism $f_Q : \mathbb{G}_1 \to \mathbb{G}_2$ by $f_Q(P) = \hat{e}(P, Q)$. If any one of these isomorphisms turns out to be invertible, then BDH is easy in $\langle \mathbb{G}_1, \mathbb{G}_2, \hat{e} \rangle$. Fortunately, an efficient algorithm for inverting f_Q for some fixed Q would imply an efficient algorithm for deciding DDH in the group $\mathbb{G}_2$. In all our examples DDH is believed to be hard in the group $\mathbb{G}_2$. Hence, all the isomorphisms $f_Q : \mathbb{G}_1 \to \mathbb{G}_2$ induced by the bilinear map are believed to be one-way functions.

4. Our IBE scheme.

We describe our scheme in stages. First we give a basic IBE scheme which is not secure against an adaptive chosen ciphertext attack. The only reason for describing the basic scheme is to make the presentation easier to follow. Our full scheme, described in section 4.2, extends the basic scheme to get security against an adaptive chosen ciphertext attack (IND-ID-CCA) in the random oracle model. In section 4.3, we relax some of the requirements on the hash functions.

The presentation in this section uses an arbitrary BDH parameter generator $\mathcal{G}$ satisfying the BDH assumption. In section 5, we describe a concrete IBE system using the Weil pairing.

4.1. BasicIdent.

To explain the basic ideas underlying our IBE system, we describe the following simple scheme, called BasicIdent. We present the scheme by describing the four algorithms: Setup, Extract, Encrypt, Decrypt. We let k be the security parameter given to the setup algorithm. We let $\mathcal{G}$ be some BDH parameter generator.

Setup: Given a security parameter $k \in \mathbb{Z}^+$, the algorithm works as follows:

Step 1: Run $\mathcal{G}$ on input k to generate a prime q, two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, and an admissible bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Choose an arbitrary generator $P \in \mathbb{G}_1$.

Step 2: Pick a random $s \in \mathbb{Z}_q^*$, and set $P_{pub} = sP$.

Step 3: Choose a cryptographic hash function $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$. Choose a cryptographic hash function $H_2 : \mathbb{G}_2 \to \{0,1\}^n$ for some n. The security analysis will view H_1, H_2 as random oracles.

The message space is $\mathcal{M} = \{0,1\}^n$. The ciphertext space is $\mathcal{C} = \mathbb{G}_1^* \times \{0,1\}^n$. The system parameters are $\mathsf{params} = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, H_1, H_2 \rangle$. The master-key is $s \in \mathbb{Z}_q^*$.

Extract: For a given string $\mathsf{ID} \in \{0,1\}^*$ the algorithm does the following: (1) it computes $Q_{\mathsf{ID}} = H_1(\mathsf{ID}) \in \mathbb{G}_1^*$, and (2) it sets the private key d_{ID} to be $d_{\mathsf{ID}} = sQ_{\mathsf{ID}}$, where s is the master key.

Encrypt: To encrypt $M \in \mathcal{M}$ under the public key ID, do the following: (1) compute $Q_{\mathsf{ID}} = H_1(\mathsf{ID}) \in \mathbb{G}_1^*$, (2) choose a random $r \in \mathbb{Z}_q^*$, and (3) set the ciphertext to be

$$C = \langle rP,\ M \oplus H_2(g_{\mathsf{ID}}^r) \rangle, \quad \text{where} \quad g_{\mathsf{ID}} = \hat{e}(Q_{\mathsf{ID}}, P_{pub}) \in \mathbb{G}_2^*.$$

Decrypt: Let $C = \langle U, V\rangle \in \mathcal{C}$ be a ciphertext encrypted using the public key ID. To decrypt C using the private key $d_{\mathsf{ID}} \in \mathbb{G}_1^*$, compute

$$V \oplus H_2(\hat{e}(d_{\mathsf{ID}}, U)) = M.$$

This completes the description of BasicIdent. We first verify consistency. When everything is computed as above, we have the following:

1. During encryption, M is bitwise exclusive-ored with the hash of g_{ID}^r.
2. During decryption, V is bitwise exclusive-ored with the hash of $\hat{e}(d_{\mathsf{ID}}, U)$.

These masks used during encryption and decryption are the same since

$$\hat{e}(d_{\mathsf{ID}}, U) = \hat{e}(sQ_{\mathsf{ID}}, rP) = \hat{e}(Q_{\mathsf{ID}}, P)^{sr} = \hat{e}(Q_{\mathsf{ID}}, P_{pub})^r = g_{\mathsf{ID}}^r.$$

Thus applying decryption after encryption produces the original message M as required. Performance considerations of BasicIdent are discussed in section 5. Note that the value of g_{ID} in algorithm Encrypt is independent of the message to be encrypted. Hence there is no need to recompute g_{ID} on subsequent encryptions to the same public key ID.

Security. Next, we study the security of this basic scheme. The following theorem shows that BasicIdent is a semantically secure IBE scheme (IND-ID-CPA) assuming BDH is hard in groups generated by $\mathcal{G}$.

THEOREM 4.1. *Suppose the hash functions H_1, H_2 are random oracles. Then* BasicIdent *is a semantically secure IBE scheme* (IND-ID-CPA) *assuming BDH is hard in groups generated by $\mathcal{G}$. Concretely, suppose there is an* IND-ID-CPA *adversary $\mathcal{A}$ that has advantage $\epsilon(k)$ against the scheme* BasicIdent. *Suppose $\mathcal{A}$ makes at most $q_E > 0$ private key extraction queries and $q_{H_2} > 0$ hash queries to H_2. Then there is an algorithm $\mathcal{B}$ that solves BDH in groups generated by $\mathcal{G}$ with advantage at least*

$$Adv_{\mathcal{G},\mathcal{B}}(k) \geq \frac{2\epsilon(k)}{e(1+q_E)\cdot q_{H_2}}.$$

Here $e \approx 2.71$ is the base of the natural logarithm. The running time of $\mathcal{B}$ is $O(time(\mathcal{A}))$.

To prove the theorem, we first define a related public key encryption scheme (not an identity-based scheme) called BasicPub. BasicPub is described by three algorithms: keygen, encrypt, decrypt.

keygen: Given a security parameter $k \in \mathbb{Z}^+$, the algorithm works as follows:

Step 1: Run $\mathcal{G}$ on input k to generate two prime order groups $\mathbb{G}_1, \mathbb{G}_2$ and a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Let q be the order of $\mathbb{G}_1, \mathbb{G}_2$. Choose an arbitrary generator $P \in \mathbb{G}_1$.

Step 2: Pick a random $s \in \mathbb{Z}_q^*$, and set $P_{pub} = sP$. Pick a random $Q_{\mathsf{ID}} \in \mathbb{G}_1^*$.

Step 3: Choose a cryptographic hash function $H_2 : \mathbb{G}_2 \to \{0,1\}^n$ for some n.

Step 4: The public key is $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, Q_{\mathsf{ID}}, H_2\rangle$. The private key is $d_{\mathsf{ID}} = sQ_{\mathsf{ID}} \in \mathbb{G}_1^*$.

encrypt: To encrypt $M \in \{0,1\}^n$, choose a random $r \in \mathbb{Z}_q^*$, and set the ciphertext to

$$C = \langle rP,\ \ M \oplus H_2(g^r)\rangle, \quad \text{where} \quad g = \hat{e}(Q_{\mathsf{ID}}, P_{pub}) \in \mathbb{G}_2^*.$$

decrypt: Let $C = \langle U, V\rangle$ be a ciphertext created using the BasicPub public key $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, Q_{\mathsf{ID}}, H_2\rangle$. To decrypt C using the private key $d_{\mathsf{ID}} \in \mathbb{G}_1^*$, compute

$$V \oplus H_2(\hat{e}(d_{\mathsf{ID}}, U)) = M.$$

This completes the description of BasicPub. We now prove Theorem 4.1 in two steps. We first show that an IND-ID-CPA attack on BasicIdent can be converted to an IND-CPA attack on BasicPub. This step shows that private key extraction queries do not help the adversary. We then show that BasicPub is IND-CPA secure if the BDH assumption holds.

LEMMA 4.2. *Let H_1 be a random oracle from $\{0,1\}^*$ to $\mathbb{G}_1^*$. Let $\mathcal{A}$ be an* IND-ID-CPA *adversary that has advantage $\epsilon(k)$ against* BasicIdent. *Suppose $\mathcal{A}$ makes at most $q_E > 0$ private key extraction queries. Then there is an* IND-CPA *adversary $\mathcal{B}$ that has advantage at least $\epsilon(k)/e(1+q_E)$ against* BasicPub. *Its running time is $O(time(\mathcal{A}))$.*

Proof. We show how to construct an IND-CPA adversary $\mathcal{B}$ that uses $\mathcal{A}$ to gain advantage $\epsilon/e(1+q_E)$ against BasicPub. The game between the challenger and the adversary $\mathcal{B}$ starts with the challenger first generating a random public key by running algorithm keygen of BasicPub. The result is a public key $K_{pub} = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, Q_{\mathsf{ID}}, H_2\rangle$ and a private key $d_{\mathsf{ID}} = sQ_{\mathsf{ID}}$. Let q be the order of $\mathbb{G}_1, \mathbb{G}_2$. The challenger gives K_{pub} to algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ is supposed to output two messages M_0 and M_1 and expects to receive back the BasicPub encryption of M_b under K_{pub}, where $b \in \{0,1\}$. Then algorithm $\mathcal{B}$ outputs its guess $b' \in \{0,1\}$ for b.

Algorithm $\mathcal{B}$ works by interacting with $\mathcal{A}$ in an IND-ID-CPA game as follows ($\mathcal{B}$ simulates the challenger).

Setup. Algorithm $\mathcal{B}$ starts by giving algorithm $\mathcal{A}$ the BasicIdent system parameters $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, H_1, H_2\rangle$. Here $q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}$, n, P, P_{pub}, H_2 are taken from K_{pub}, and H_1 is a random oracle controlled by $\mathcal{B}$ as described below.

***H_1*-queries.** At any time algorithm $\mathcal{A}$ can query the random oracle H_1. To respond to these queries algorithm $\mathcal{B}$ maintains a list of tuples $\langle \mathsf{ID}_j, Q_j, b_j, c_j\rangle$ as explained below. We refer to this list as the H_1^{list}. The list is initially empty. When $\mathcal{A}$ queries the oracle H_1 at a point ID_i, algorithm $\mathcal{B}$ responds as follows:

1. If the query ID_i already appears on the H_1^{list} in a tuple $\langle \mathsf{ID}_i, Q_i, b_i, c_i\rangle$, then Algorithm $\mathcal{B}$ responds with $H_1(\mathsf{ID}_i) = Q_i \in \mathbb{G}_1^*$.
2. Otherwise, $\mathcal{B}$ generates a random $coin \in \{0,1\}$ so that $\Pr[coin = 0] = \delta$ for some δ that will be determined later.
3. Algorithm $\mathcal{B}$ picks a random $b \in \mathbb{Z}_q^*$.
   If $coin = 0$, compute $Q_i = bP \in \mathbb{G}_1^*$. If $coin = 1$, compute $Q_i = bQ_{\mathsf{ID}} \in \mathbb{G}_1^*$.
4. Algorithm $\mathcal{B}$ adds the tuple $\langle \mathsf{ID}_i, Q_i, b, coin\rangle$ to the H_1^{list} and responds to $\mathcal{A}$ with $H_1(\mathsf{ID}_i) = Q_i$.

Note that either way Q_i is uniform in $\mathbb{G}_1^*$ and is independent of $\mathcal{A}$'s current view as required.

Phase 1. Let ID_i be a private key extraction query issued by algorithm $\mathcal{A}$. Algorithm $\mathcal{B}$ responds to this query as follows:

1. Run the above algorithm for responding to H_1-queries to obtain a $Q_i \in \mathbb{G}_1^*$ such that $H_1(\mathsf{ID}_i) = Q_i$. Let $\langle \mathsf{ID}_i, Q_i, b_i, coin_i\rangle$ be the corresponding tuple on the H_1^{list}. If $coin_i = 1$, then $\mathcal{B}$ reports failure and terminates. The attack on BasicPub failed.
2. We know $coin_i = 0$ and hence $Q_i = b_iP$. Define $d_i = b_iP_{pub} \in \mathbb{G}_1^*$. Observe that $d_i = sQ_i$, and therefore d_i is the private key associated to the public key ID_i. Give d_i to algorithm $\mathcal{A}$.

Challenge. Once algorithm $\mathcal{A}$ decides that Phase 1 is over, it outputs a public key ID_{ch} and two messages M_0, M_1 on which it wishes to be challenged. Algorithm $\mathcal{B}$ responds as follows:

1. Algorithm $\mathcal{B}$ gives the challenger M_0, M_1 as the messages that it wishes to be challenged on. The challenger responds with a BasicPub ciphertext $C = \langle U, V\rangle$ such that C is the encryption of M_c for a random $c \in \{0,1\}$.
2. Next, $\mathcal{B}$ runs the algorithm for responding to H_1-queries to obtain a $Q \in \mathbb{G}_1^*$ such that $H_1(\mathsf{ID}_{ch}) = Q$. Let $\langle \mathsf{ID}_{ch}, Q, b, coin\rangle$ be the corresponding tuple on the H_1^{list}. If $coin = 0$, then $\mathcal{B}$ reports failure and terminates. The attack on BasicPub failed.
3. We know $coin = 1$, and therefore $Q = bQ_{\mathsf{ID}}$. Recall that when $C = \langle U, V\rangle$, we have $U \in \mathbb{G}_1^*$. Set $C' = \langle b^{-1}U,\ V\rangle$, where b^{-1} is the inverse of $b \bmod q$. Algorithm $\mathcal{B}$ responds to $\mathcal{A}$ with the challenge C'. Note that C' is a BasicIdent encryption of M_c under the public key ID_{ch} as required. To see this, first observe that, since $H_1(\mathsf{ID}_{ch}) = Q$, the private key corresponding to ID_{ch} is $d_{ch} = sQ$. Second, observe that

$$\hat{e}(b^{-1}U, d_{ch}) = \hat{e}(b^{-1}U, sQ) = \hat{e}(U, sb^{-1}Q) = \hat{e}(U, sQ_{\mathsf{ID}}) = \hat{e}(U, d_{\mathsf{ID}}).$$

   Hence the BasicIdent decryption of C' using d_{ch} is the same as the BasicPub decryption of C using d_{ID}.

Phase 2. Algorithm $\mathcal{B}$ responds to private key extraction queries in the same way it did in Phase 1.

Guess. Eventually algorithm $\mathcal{A}$ produces a guess c' for c. Algorithm $\mathcal{B}$ outputs c' as its guess for c.

Claim. If algorithm $\mathcal{B}$ does not abort during the simulation, then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Furthermore, if $\mathcal{B}$ does not abort, then $|\Pr[c = c'] - \frac{1}{2}| \geq \epsilon$. The probability is over the random bits used by $\mathcal{A}, \mathcal{B}$ and the challenger.

Proof of claim. The responses to H_1-queries are as in the real attack since each response is uniformly and independently distributed in $\mathbb{G}_1^*$. All responses to private key extraction queries are valid. Finally, the challenge ciphertext C' given to $\mathcal{A}$ is the BasicIdent encryption of M_c for some random $c \in \{0,1\}$. Therefore, by definition of algorithm $\mathcal{A}$, we have that $|\Pr[c = c'] - \frac{1}{2}| \geq \epsilon$. □

To complete the proof of Lemma 4.2, it remains to calculate the probability that algorithm $\mathcal{B}$ aborts during the simulation. Suppose $\mathcal{A}$ makes a total of q_E private key extraction queries. Then the probability that $\mathcal{B}$ does not abort in Phases 1 or 2 is δ^{q_E}. The probability that it does not abort during the challenge step is $1 - \delta$. Therefore, the probability that $\mathcal{B}$ does not abort during the simulation is $\delta^{q_E}(1 - \delta)$. This value is maximized at $\delta_{opt} = 1 - 1/(q_E + 1)$. Using δ_{opt}, the probability that $\mathcal{B}$ does not abort is at least $1/e(1 + q_E)$. This shows that $\mathcal{B}$'s advantage is at least $\epsilon/e(1 + q_E)$ as required. □

The analysis used in the proof of Lemma 4.2 uses a similar technique to Coron's analysis of the full domain hash signature scheme [9]. Next, we show that BasicPub is a semantically secure public key system if the BDH assumption holds.

LEMMA 4.3. *Let H_2 be a random oracle from $\mathbb{G}_2$ to $\{0,1\}^n$. Let $\mathcal{A}$ be an* IND-CPA *adversary that has advantage $\epsilon(k)$ against* BasicPub. *Suppose $\mathcal{A}$ makes a total of $q_{H_2} > 0$ queries to H_2. Then there is an algorithm $\mathcal{B}$ that solves the BDH problem for $\mathcal{G}$ with advantage at least $2\epsilon(k)/q_{H_2}$ and a running time $O(time(\mathcal{A}))$.*

Proof. Algorithm $\mathcal{B}$ is given as input the BDH parameters $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}\rangle$ produced by $\mathcal{G}$ and a random instance $\langle P, aP, bP, cP\rangle = \langle P, P_1, P_2, P_3\rangle$ of the BDH problem for these parameters; i.e., P is random in $\mathbb{G}_1^*$ and a, b, c are random in $\mathbb{Z}_q^*$, where q is

the order of $\mathbb{G}_1, \mathbb{G}_2$. Let $D = \hat{e}(P,P)^{abc} \in \mathbb{G}_2$ be the solution to this BDH problem. Algorithm $\mathcal{B}$ finds D by interacting with $\mathcal{A}$ as follows.

Setup. Algorithm $\mathcal{B}$ sets $P_{pub} = P_1$ and $Q_{\mathsf{ID}} = P_2$ and creates the BasicPub public key $K_{pub} = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, Q_{\mathsf{ID}}, H_2 \rangle$. Here H_2 is a random oracle controlled by $\mathcal{B}$ as described below. Algorithm $\mathcal{B}$ gives $\mathcal{A}$ the BasicPub public key K_{pub}. Observe that the (unknown) private key associated to K_{pub} is $d_{\mathsf{ID}} = aQ_{\mathsf{ID}} = abP$.

***H_2*-queries.** At any time, algorithm $\mathcal{A}$ may issue queries to the random oracle H_2. To respond to these queries, $\mathcal{B}$ maintains a list of tuples called the H_2^{list}. Each entry in the list is a tuple of the form $\langle X_j, H_j \rangle$. Initially the list is empty. To respond to query X_i, algorithm $\mathcal{B}$ does the following:

1. If the query X_i already appears on the H_2^{list} in a tuple $\langle X_i, H_i \rangle$, then respond with $H_2(X_i) = H_i$.
2. Otherwise, $\mathcal{B}$ just picks a random string $H_i \in \{0,1\}^n$ and adds the tuple $\langle X_i, H_i \rangle$ to the H_2^{list}. It responds to $\mathcal{A}$ with $H_2(X_i) = H_i$.

Challenge. Algorithm $\mathcal{A}$ outputs two messages M_0, M_1 on which it wishes to be challenged. Algorithm $\mathcal{B}$ picks a random string $R \in \{0,1\}^n$ and defines C to be the ciphertext $C = \langle P_3, R \rangle$. Algorithm $\mathcal{B}$ gives C as the challenge to $\mathcal{A}$. Observe that, by definition, the decryption of C is $R \oplus H_2(\hat{e}(P_3, d_{\mathsf{ID}})) = R \oplus H_2(D)$.

Guess. Algorithm $\mathcal{A}$ outputs its guess $c' \in \{0,1\}$. At this point, $\mathcal{B}$ picks a random tuple $\langle X_j, H_j \rangle$ from the H_2^{list} and outputs X_j as the solution to the given instance of BDH.

Algorithm $\mathcal{B}$ is simulating a real attack environment for algorithm $\mathcal{A}$ (it simulates the challenger and the oracle for H_2). We show that algorithm $\mathcal{B}$ outputs the correct answer D with probability at least $2\epsilon/q_{H_2}$ as required. The proof is based on comparing $\mathcal{A}$'s behavior in the simulation to its behavior in a real IND-CPA attack game (against a real challenger and a real random oracle for H_2).

Let $\mathcal{H}$ be the event that algorithm $\mathcal{A}$ issues a query for $H_2(D)$ at some point during the simulation above. (This implies that at the end of the simulation D appears in some tuple on the H_2^{list}.) We show that $\Pr[\mathcal{H}] \geq 2\epsilon$. This will prove that algorithm $\mathcal{B}$ outputs D with probability at least $2\epsilon/q_{H_2}$. We also study event $\mathcal{H}$ in the real attack game, namely, the event that $\mathcal{A}$ issues a query for $H_2(D)$ when communicating with a real challenger and a real random oracle for H_2.

Claim 1. $\Pr[\mathcal{H}]$ in the simulation above is equal to $\Pr[\mathcal{H}]$ in the real attack.

Proof of claim. Let $\mathcal{H}_\ell$ be the event that $\mathcal{A}$ makes a query for $H_2(D)$ in one of its first ℓ queries to the H_2 oracle. We prove by induction on ℓ that $\Pr[\mathcal{H}_\ell]$ in the real attack is equal to $\Pr[\mathcal{H}_\ell]$ in the simulation for all $\ell \geq 0$. Clearly, $\Pr[\mathcal{H}_0] = 0$ in both the simulation and in the real attack. Now suppose that, for some $\ell > 0$, we have that $\Pr[\mathcal{H}_{\ell-1}]$ in the simulation is equal to $\Pr[\mathcal{H}_{\ell-1}]$ in the real attack. We show that the same holds for $\mathcal{H}_\ell$. We know that

$$\begin{aligned} \Pr[\mathcal{H}_\ell] &= \Pr[\mathcal{H}_\ell \mid \mathcal{H}_{\ell-1}] \Pr[\mathcal{H}_{\ell-1}] + \Pr[\mathcal{H}_\ell \mid \neg\mathcal{H}_{\ell-1}] \Pr[\neg\mathcal{H}_{\ell-1}] \\ &= \Pr[\mathcal{H}_{\ell-1}] + \Pr[\mathcal{H}_\ell \mid \neg\mathcal{H}_{\ell-1}] \Pr[\neg\mathcal{H}_{\ell-1}]. \end{aligned} \tag{4.1}$$

We argue that $\Pr[\mathcal{H}_\ell \mid \neg\mathcal{H}_{\ell-1}]$ in the simulation is equal to $\Pr[\mathcal{H}_\ell \mid \neg\mathcal{H}_{\ell-1}]$ in the real attack. To see this, observe that as long as $\mathcal{A}$ does not issue a query for $H_2(D)$, its view during the simulation is identical to its view in the real attack (against a real challenger and a real random oracle for H_2). Indeed, the public key and the challenge are distributed as in the real attack. Similarly, all responses to H_2-queries are uniform and independent in $\{0,1\}^n$. Therefore, $\Pr[\mathcal{H}_\ell \mid \neg\mathcal{H}_{\ell-1}]$ in the simulation is equal to

$\Pr[\mathcal{H}_\ell \mid \neg\mathcal{H}_{\ell-1}]$ in the real attack. It follows by (4.1) and the inductive hypothesis that $\Pr[\mathcal{H}_\ell]$ in the real attack is equal to $\Pr[\mathcal{H}_\ell]$ in the simulation. By induction on ℓ, we obtain that $\Pr[\mathcal{H}]$ in the real attack is equal to $\Pr[\mathcal{H}]$ in the simulation. □

Claim 2. In the real attack, we have $\Pr[\mathcal{H}] \geq 2\epsilon$.

Proof of claim. In the real attack, if $\mathcal{A}$ never issues a query for $H_2(D)$, then the decryption of C is independent of $\mathcal{A}$'s view (since $H_2(D)$ is independent of $\mathcal{A}$'s view). Therefore, in the real attack, $\Pr[c = c' \mid \neg\mathcal{H}] = 1/2$. By definition of $\mathcal{A}$, we know that in the real attack $|\Pr[c = c'] - 1/2| \geq \epsilon$. We show that these two facts imply that $\Pr[\mathcal{H}] \geq 2\epsilon$. To do so, we first derive simple upper and lower bounds on $\Pr[c = c']$:

$$\begin{aligned}\Pr[c = c'] &= \Pr[c = c' | \neg\mathcal{H}] \Pr[\neg\mathcal{H}] + \Pr[c = c' | \mathcal{H}] \Pr[\mathcal{H}] \\ &\leq \Pr[c = c' | \neg\mathcal{H}] \Pr[\neg\mathcal{H}] + \Pr[\mathcal{H}] = \frac{1}{2}\Pr[\neg\mathcal{H}] + \Pr[\mathcal{H}] = \frac{1}{2} + \frac{1}{2}\Pr[\mathcal{H}], \\ \Pr[c = c'] &\geq \Pr[c = c' | \neg\mathcal{H}] \Pr[\neg\mathcal{H}] = \frac{1}{2} - \frac{1}{2}\Pr[\mathcal{H}].\end{aligned}$$

It follows that $\epsilon \leq |\Pr[c = c'] - 1/2| \leq \frac{1}{2}\Pr[\mathcal{H}]$. Therefore, in the real attack, $\Pr[\mathcal{H}] \geq 2\epsilon$. □

To complete the proof of Lemma 4.3, observe that by Claims 1 and 2 we know that $\Pr[\mathcal{H}] \geq 2\epsilon$ in the simulation above. Hence, at the end of the simulation, D appears in some tuple on the H_2^{list} with probability at least 2ϵ. It follows that $\mathcal{B}$ produces the correct answer with probability at least $2\epsilon/q_{H_2}$ as required. □

We note that one can slightly vary the reduction in the proof above to obtain different bounds. For example, in the "Guess" step above, one can avoid having to pick a random element from the H_2^{list} by using the random self-reduction of the BDH problem. This requires running algorithm $\mathcal{A}$ multiple times (as in Theorem 7 of [42]). The success probability for solving the given BDH problem increases at the cost of also increasing the running time.

Proof of Theorem 4.1. The theorem follows from Lemmas 4.2 and 4.3. Composing both reductions shows that an IND-ID-CPA adversary on BasicIdent with advantage $\epsilon(k)$ gives a BDH algorithm for $\mathcal{G}$ with advantage at least $2\epsilon(k)/e(1 + q_E)q_{H_2}$, as required. □

4.2. IBE with chosen ciphertext security. We use a technique due to Fujisaki and Okamoto [16] to convert the BasicIdent scheme of the previous section into a chosen ciphertext secure IBE system (in the sense of section 2) in the random oracle model. Let $\mathcal{E}$ be a probabilistic public key encryption scheme. We denote by $\mathcal{E}_{pk}(M; r)$ the encryption of M using the random bits r under the public key pk. Fujisaki and Okamoto define the hybrid scheme $\mathcal{E}^{hy}$ as

$$\mathcal{E}^{hy}_{pk}(M) \;=\; \langle\, \mathcal{E}_{pk}(\sigma; H_3(\sigma, M)),\;\; H_4(\sigma) \oplus M \,\rangle.$$

Here σ is generated at random, and H_3, H_4 are cryptographic hash functions. Fujisaki and Okamoto show that if $\mathcal{E}$ is a one-way encryption scheme, then $\mathcal{E}^{hy}$ is a chosen ciphertext secure system (IND-CCA) in the random oracle model (assuming $\mathcal{E}_{pk}$ satisfies some natural constraints). We note that semantic security implies one-way encryption, and hence the Fujisaki–Okamoto result also applies if $\mathcal{E}$ is semantically secure (IND-CPA).

We apply the Fujisaki–Okamoto transformation to BasicIdent and show that the resulting IBE system is IND-ID-CCA secure. We obtain the following IBE scheme which we call FullIdent. Recall that n is the length of the message to be encrypted.

Setup. As in the BasicIdent scheme. In addition, we pick a hash function $H_3 : \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}_q^*$ and a hash function $H_4 : \{0,1\}^n \to \{0,1\}^n$.

Extract. As in the BasicIdent scheme.

Encrypt. To encrypt $M \in \{0,1\}^n$ under the public key ID, do the following: (1) compute $Q_{\mathsf{ID}} = H_1(\mathsf{ID}) \in \mathbb{G}_1^*$, (2) choose a random $\sigma \in \{0,1\}^n$, (3) set $r = H_3(\sigma, M)$, and (4) set the ciphertext to be

$$C = \langle rP,\ \sigma \oplus H_2(g_{\mathsf{ID}}^r),\ M \oplus H_4(\sigma)\rangle, \quad \text{where} \quad g_{\mathsf{ID}} = \hat{e}(Q_{\mathsf{ID}}, P_{pub}) \in \mathbb{G}_2.$$

Decrypt. Let $C = \langle U, V, W\rangle$ be a ciphertext encrypted using the public key ID. If $U \notin \mathbb{G}_1^*$, reject the ciphertext. To decrypt C using the private key $d_{\mathsf{ID}} \in \mathbb{G}_1^*$, do:

1. Compute $V \oplus H_2(\hat{e}(d_{\mathsf{ID}}, U)) = \sigma$.
2. Compute $W \oplus H_4(\sigma) = M$.
3. Set $r = H_3(\sigma, M)$. Test that $U = rP$. If not, reject the ciphertext.
4. Output M as the decryption of C.

This completes the description of FullIdent. Note that M is encrypted as $W = M \oplus H_4(\sigma)$. This can be replaced by $W = E_{H_4(\sigma)}(M)$, where E is a semantically secure symmetric encryption scheme (see [16]).

Security. The following theorem shows that FullIdent is a chosen ciphertext secure IBE (i.e., IND-ID-CCA), assuming BDH is hard in groups generated by $\mathcal{G}$.

THEOREM 4.4. *Let the hash functions H_1, H_2, H_3, H_4 be random oracles. Then* FullIdent *is a chosen ciphertext secure IBE* (IND-ID-CCA)*, assuming BDH is hard in groups generated by $\mathcal{G}$.*

Concretely, suppose there is an IND-ID-CCA *adversary $\mathcal{A}$ that has advantage $\epsilon(k)$ against the scheme* FullIdent *and $\mathcal{A}$ runs in time at most $t(k)$. Suppose $\mathcal{A}$ makes at most q_E extraction queries, at most q_D decryption queries, and at most $q_{H_2}, q_{H_3}, q_{H_4}$ queries to the hash functions H_2, H_3, H_4, respectively. Then there is a BDH algorithm $\mathcal{B}$ for $\mathcal{G}$ with running time $t_1(k)$, where*

$$Adv_{\mathcal{G},\mathcal{B}}(k) \geq 2FO_{adv}\left(\tfrac{\epsilon(k)}{e(1+q_E+q_D)},\ q_{H_4},\ q_{H_3},\ q_D\right)/q_{H_2},$$
$$t_1(k) \leq FO_{time}(t(k), q_{H_4}, q_{H_3}),$$

where the functions FO_{time} and FO_{adv} are as defined in Theorem 4.5.

The proof of Theorem 4.4 is based on the following result of Fujisaki and Okamoto (Theorem 14 in [16]). Let BasicPub^{hy} be the result of applying the Fujisaki–Okamoto transformation to BasicPub.

THEOREM 4.5 (Fujisaki–Okamoto). *Suppose $\mathcal{A}$ is an* IND-CCA *adversary that achieves advantage $\epsilon(k)$ when attacking* BasicPub^{hy}*. Suppose $\mathcal{A}$ has running time $t(k)$, makes at most q_D decryption queries, and makes at most q_{H_3}, q_{H_4} queries to the hash functions H_3, H_4, respectively. Then there is an* IND-CPA *adversary $\mathcal{B}$ against* BasicPub *with running time $t_1(k)$ and advantage $\epsilon_1(k)$, where*

$$\epsilon_1(k) \geq FO_{adv}(\epsilon(k), q_{H_4}, q_{H_3}, q_D) = \frac{1}{2(q_{H_4}+q_{H_3})}\left[(\epsilon(k)+1)(1-2/q)^{q_D} - 1\right],$$
$$t_1(k) \leq FO_{time}(t(k), q_{H_4}, q_{H_3}) = t(k) + O((q_{H_4}+q_{H_3})\cdot n).$$

Here q is the size of the groups $\mathbb{G}_1, \mathbb{G}_2$, and n is the length of σ.

In fact, Fujisaki and Okamoto prove a stronger result: Under the hypothesis of Theorem 4.5, BasicPub^{hy} would not even be a one-way encryption scheme. For our

purposes, the result in Theorem 4.5 is sufficient. To prove Theorem 4.4, we also need the following lemma to translate between an IND-ID-CCA chosen ciphertext attack on FullIdent and an IND-CCA chosen ciphertext attack on BasicPub^{hy}.

LEMMA 4.6. *Let $\mathcal{A}$ be an* IND-ID-CCA *adversary that has advantage $\epsilon(k)$ against* FullIdent. *Suppose $\mathcal{A}$ makes at most $q_E > 0$ private key extraction queries and at most q_D decryption queries. Then there is an* IND-CCA *adversary $\mathcal{B}$ that has advantage at least $\frac{\epsilon(k)}{e(1+q_E+q_D)}$ against* BasicPub^{hy}. *Its running time is $O(time(\mathcal{A}))$.*

Proof. We construct an IND-CCA adversary $\mathcal{B}$ that uses $\mathcal{A}$ to gain advantage $\epsilon/e(1 + q_E + q_D)$ against BasicPub^{hy}. The game between the challenger and the adversary $\mathcal{B}$ starts with the challenger first generating a random public key by running algorithm keygen of BasicPub^{hy}. The result is a public key

$$K_{pub} = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, Q_{\mathsf{ID}}, H_2, H_3, H_4 \rangle$$

and a private key $d_{\mathsf{ID}} = sQ_{\mathsf{ID}}$. The challenger gives K_{pub} to algorithm $\mathcal{B}$.

Algorithm $\mathcal{B}$ mounts an IND-CCA attack on the key K_{pub} using the help of algorithm $\mathcal{A}$. Algorithm $\mathcal{B}$ interacts with $\mathcal{A}$ as follows.

Setup. Same as in Lemma 4.2 (with H_3, H_4 included in the system parameters given to $\mathcal{A}$).

***H_1*-queries.** These queries are handled as in Lemma 4.2.

Phase 1. Private key queries. These are handled as in Lemma 4.2.

Phase 1. Decryption queries. Let $\langle \mathsf{ID}_i, C_i \rangle$ be a decryption query issued by algorithm $\mathcal{A}$. Let $C_i = \langle U_i, V_i, W_i \rangle$. Algorithm $\mathcal{B}$ responds to this query as follows:

1. Run the above algorithm for responding to H_1-queries to obtain a $Q_i \in \mathbb{G}_1^*$ such that $H_1(\mathsf{ID}_i) = Q_i$. Let $\langle \mathsf{ID}_i, Q_i, b_i, coin_i \rangle$ be the corresponding tuple on the H_1^{list}.
2. Suppose $coin_i = 0$. In this case, run the algorithm for responding to private key queries to obtain the private key for the public key ID_i. Then use the private key to respond to the decryption query.
3. Suppose $coin_i = 1$. Then $Q_i = b_i Q_{\mathsf{ID}}$.
   – Recall that $U_i \in \mathbb{G}_1$. Set $C_i' = \langle b_i U_i,\ V_i,\ W_i \rangle$. Let $d_i = sQ_i$ be the (unknown) FullIdent private key corresponding to ID_i. Then the FullIdent decryption of C_i using d_i is the same as the BasicPub^{hy} decryption of C_i' using d_{ID}. To see this, observe that

$$\hat{e}(b_i U_i, d_{\mathsf{ID}}) = \hat{e}(b_i U_i, sQ_{\mathsf{ID}}) = \hat{e}(U_i, sb_i Q_{\mathsf{ID}}) = \hat{e}(U_i, sQ_i) = \hat{e}(U_i, d_i).$$

   – Relay the decryption query $\langle C_i' \rangle$ to the challenger and relay the challenger's response back to $\mathcal{A}$.

Challenge. Once algorithm $\mathcal{A}$ decides that Phase 1 is over, it outputs a public key ID_{ch} and two messages M_0, M_1 on which it wishes to be challenged. Algorithm $\mathcal{B}$ responds as follows:

1. Algorithm $\mathcal{B}$ gives the challenger M_0, M_1 as the messages that it wishes to be challenged on. The challenger responds with a BasicPub^{hy} ciphertext $C = \langle U, V, W \rangle$ such that C is the encryption of M_c for a random $c \in \{0, 1\}$.
2. Next, $\mathcal{B}$ runs the algorithm for responding to H_1-queries to obtain a $Q \in \mathbb{G}_1^*$ such that $H_1(\mathsf{ID}_{ch}) = Q$. Let $\langle \mathsf{ID}_{ch}, Q, b, coin \rangle$ be the corresponding tuple on the H_1^{list}. If $coin = 0$, then $\mathcal{B}$ reports failure and terminates. The attack on BasicPub^{hy} failed.

3. We know $coin = 1$ and therefore $Q = bQ_{\mathsf{ID}}$. Recall that when $C = \langle U, V, W\rangle$, we have $U \in \mathbb{G}_1^*$. Set $C' = \langle b^{-1}U,\ V,\ W\rangle$, where b^{-1} is the inverse of b mod q. Algorithm $\mathcal{B}$ responds to $\mathcal{A}$ with the challenge C'. Note that, as in the proof of Lemma 4.2, C' is a FullIdent encryption of M_c under the public key ID_{ch} as required.

Phase 2. Private key queries. Algorithm $\mathcal{B}$ responds to private key extraction queries in the same way it did in Phase 1.

Phase 2. Decryption queries. Algorithm $\mathcal{B}$ responds to decryption queries in the same way it did in Phase 1. However, if the resulting decryption query relayed to the challenger is equal to the challenge ciphertext $C = \langle U, V, W\rangle$, then $\mathcal{B}$ reports failure and terminates. The attack on $\mathsf{BasicPub}^{hy}$ failed.

Guess. Eventually algorithm $\mathcal{A}$ produces a guess c' for c. Algorithm $\mathcal{B}$ outputs c' as its guess for c.

Claim. If algorithm $\mathcal{B}$ does not abort during the simulation, then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Furthermore, if $\mathcal{B}$ does not abort, then $|\Pr[c = c'] - \frac{1}{2}| \geq \epsilon$. The probability is over the random bits used by $\mathcal{A}, \mathcal{B}$ and the challenger.

Proof of claim. The responses to H_1-queries are as in the real attack since each response is uniformly and independently distributed in $\mathbb{G}_1^*$. All responses to private key extraction queries and decryption queries are valid. Finally, the challenge ciphertext C' given to $\mathcal{A}$ is the FullIdent encryption of M_c for some random $c \in \{0, 1\}$. Therefore, by definition of algorithm $\mathcal{A}$, we have that $|\Pr[c = c'] - \frac{1}{2}| \geq \epsilon$. □

It remains to bound the probability that algorithm $\mathcal{B}$ aborts during the simulation. The algorithm could abort for three reasons: (1) a bad private key query from $\mathcal{A}$ during Phase 1 or 2, (2) $\mathcal{A}$ chooses a bad ID_{ch} to be challenged on, or (3) a bad decryption query from $\mathcal{A}$ during phase 2. We define three corresponding events.

- $\mathcal{E}_1$ is the event that $\mathcal{A}$ issues a private key query during Phase 1 or 2 that causes algorithm $\mathcal{B}$ to abort.
- $\mathcal{E}_2$ is the event that $\mathcal{A}$ chooses a public key ID_{ch} to be challenged on that causes algorithm $\mathcal{B}$ to abort.
- $\mathcal{E}_3$ is the event that, during Phase 2 of the simulation, Algorithm $\mathcal{A}$ issues a decryption query $\langle \mathsf{ID}_i, C_i\rangle$ so that the decryption query that $\mathcal{B}$ would relay to the $\mathsf{BasicPub}^{hy}$ challenger is equal to C. Recall that $C = \langle U, V, W\rangle$ is the challenge ciphertext from the $\mathsf{BasicPub}^{hy}$ challenger.

Claim. $\Pr[\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_2 \wedge \neg\mathcal{E}_3] \geq \delta^{q_E+q_D}(1-\delta)$.

Proof of claim. We prove the claim by induction on the maximum number of queries $q_E + q_D$ made by the adversary. Let $i = q_E + q_D$, and let $\mathcal{E}^{0\ldots i}$ be the event that $\mathcal{E}_1 \vee \mathcal{E}_3$ happens after $\mathcal{A}$ issues at most i queries. Similarly, let $\mathcal{E}^i$ be the event that $\mathcal{E}_1 \vee \mathcal{E}_3$ happens for the first time when $\mathcal{A}$ issues the ith query. We prove by induction on i that $\Pr[\neg\mathcal{E}^{0\ldots i} \mid \neg\mathcal{E}_2] \geq \delta^i$. The claim follows because $\Pr[\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_2 \wedge \neg\mathcal{E}_3] = \Pr[\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_3 \mid \neg\mathcal{E}_2]\Pr[\neg\mathcal{E}_2] \geq \Pr[\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_3 \mid \neg\mathcal{E}_2](1-\delta)$.

For $i = 0$, the claim is trivial since by definition $\Pr[\neg\mathcal{E}^{0\ldots 0}] = 1$. Now, suppose the claim holds for $i - 1$. Then

$$\begin{aligned}\Pr[\neg\mathcal{E}^{0\ldots i} \mid \neg\mathcal{E}_2] &= \Pr[\neg\mathcal{E}^{0\ldots i} \mid \neg\mathcal{E}^{0\ldots i-1} \wedge \neg\mathcal{E}_2]\Pr[\neg\mathcal{E}^{0\ldots i-1} \mid \neg\mathcal{E}_2] \\ &= \Pr[\neg\mathcal{E}^{i} \mid \neg\mathcal{E}^{0\ldots i-1} \wedge \neg\mathcal{E}_2]\Pr[\neg\mathcal{E}^{0\ldots i-1} \mid \neg\mathcal{E}_2] \\ &\geq \Pr[\neg\mathcal{E}^{i} \mid \neg\mathcal{E}^{0\ldots i-1} \wedge \neg\mathcal{E}_2]\delta^{i-1}.\end{aligned}$$

Hence it suffices to bound $q_i = \Pr[\neg\mathcal{E}^i \mid \neg\mathcal{E}^{0\ldots i-1} \wedge \neg\mathcal{E}_2]$. In other words, we bound the probability that the ith query does not cause $\mathcal{E}^i$ to happen given that the first

$i-1$ queries did not and given that $\mathcal{E}_2$ does not occur. Consider the ith query issued by $\mathcal{A}$ during the simulation. The query is either a private key query for $\langle \mathsf{ID}_i \rangle$ or a decryption query for $\langle \mathsf{ID}_i, C_i \rangle$, where $C_i = \langle U_i, V_i, W_i \rangle$. If the query is a decryption query, we assume it takes place during Phase 2 since otherwise it has no effect on $\mathcal{E}_3$.

Let $H_1(\mathsf{ID}_i) = Q_i$, and let $\langle \mathsf{ID}_i, Q_i, b_i, coin_i \rangle$ be the corresponding tuple on the H_1^{list}. Recall that when $coin_i = 0$, the query cannot cause event $\mathcal{E}_1$ to happen. Similarly, when $coin_i = 0$, the query cannot cause event $\mathcal{E}_3$ to happen since in this case $\mathcal{B}$ does not relay a decryption query to the $\mathsf{BasicPub}^{hy}$ challenger. We use these facts to bound q_i. There are four cases to consider. In the first three cases, we assume ID_i is not equal to the public key ID_{ch} on which $\mathcal{A}$ is being challenged.

Case 1. The ith query is the first time $\mathcal{A}$ issues a query containing ID_i. In this case, $\Pr[coin_i = 0] = \delta$, and hence $q_i \geq \delta$.

Case 2. The public key ID_i appeared in a previous private key query. Since by assumption this earlier private key query did not cause $\mathcal{E}^{0\ldots i-1}$ to happen, we know that $coin_i = 0$. Hence we have $q_i = 1$.

Case 3. The public key ID_i appeared in a previous decryption query. Since by assumption this earlier decryption query did not cause event $\mathcal{E}^{0\ldots i-1}$ to happen, we have that either $coin_i = 0$ or $coin_i$ is independent of $\mathcal{A}$'s current view. Either way, we have that $q_i \geq \delta$.

Case 4. The public key ID_i is equal to the public key ID_{ch} on which $\mathcal{A}$ is being challenged. Then, by definition, the ith query cannot be a private key query. Therefore, it must be a decryption query $\langle \mathsf{ID}_i, C_i \rangle$. Furthermore, since $\mathcal{E}_2$ did not happen, we know that $coin_i = 1$, and hence $\mathcal{B}$ will relay a decryption query C_i' to the $\mathsf{BasicPub}^{hy}$ challenger. Let C' be the challenge ciphertext given to $\mathcal{A}$. By definition we know that $C_i \neq C'$. It follows that $C_i' \neq C$. Therefore, this query cannot cause event $\mathcal{E}_3$ to happen. Hence in this case $q_i = 1$.

To summarize, we see that whatever the ith query is, we have that $q_i \geq \delta$. Therefore, we have that $\Pr[\neg\mathcal{E}^{0\ldots i} \mid \neg\mathcal{E}_2] \geq \delta^i$ as required. The claim now follows by setting $i = q_E + q_D$. □

To conclude the proof of Lemma 4.6, it remains to optimize the choice of δ. Since $\Pr[\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_2 \wedge \neg\mathcal{E}_3] \geq \delta^{q_E+q_D}(1-\delta)$, the success probability is maximized at $\delta_{opt} = 1 - 1/(q_E + q_D + 1)$. Using δ_{opt}, the probability that $\mathcal{B}$ does not abort is at least $\frac{1}{e(1+q_E+q_D)}$. This shows that $\mathcal{B}$'s advantage is at least $\epsilon/e(1 + q_E + q_D)$ as required. □

Proof of Theorem 4.4. By Lemma 4.6, an IND-ID-CCA adversary on FullIdent implies an IND-CCA adversary on $\mathsf{BasicPub}^{hy}$. By Theorem 4.5, an IND-CCA adversary on $\mathsf{BasicPub}^{hy}$ implies an IND-CPA adversary on BasicPub. By Lemma 4.3, an IND-CPA adversary on BasicPub implies an algorithm for BDH. Composing all these reductions gives the required bounds. □

4.3. Relaxing the hashing requirements. Recall that the IBE system of section 4.2 uses a hash function $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$. The concrete IBE system presented in the next section uses $\mathbb{G}_1$ as a subgroup of the group of points on an elliptic curve. In practice, it is difficult to build hash functions that hash directly onto such groups. We therefore show how to relax the requirement of hashing directly onto $\mathbb{G}_1^*$. Rather than hash onto $\mathbb{G}_1^*$, we hash onto some set $A \subseteq \{0,1\}^*$ and then use a deterministic encoding function to map A onto $\mathbb{G}_1^*$.

Admissible encodings. Let $\mathbb{G}_1$ be a group, and let $A \in \{0,1\}^*$ be a finite set. We say that an encoding function $L : A \to \mathbb{G}_1^*$ is *admissible* if it satisfies the following

properties.

1. *Computable.* There is an efficient deterministic algorithm to compute $L(x)$ for any $x \in A$.
2. ℓ-*to*-1. For any $y \in \mathbb{G}_1^*$, the preimage of y under L has size exactly ℓ. In other words, $|L^{-1}(y)| = \ell$ for all $y \in \mathbb{G}_1^*$. Note that this implies that $|A| = \ell \cdot |\mathbb{G}_1^*|$.
3. *Samplable.* There is an efficient randomized algorithm $\mathcal{L}_S$ such that $\mathcal{L}_S(y)$ induces a uniform distribution on $L^{-1}(y)$ for any $y \in \mathbb{G}_1^*$. In other words, $\mathcal{L}_S(y)$ is a uniform random element in $L^{-1}(y)$.

We slightly modify FullIdent to obtain an IND-ID-CCA secure IBE system where H_1 is replaced by a hash function into some set A. Since the change is so minor, we refer to this new scheme as FullIdent'.

Setup. As in the FullIdent scheme. The only difference is that H_1 is replaced by a hash function $H_1' : \{0,1\}^* \to A$. The system parameters also include a description of an admissible encoding function $L : A \to \mathbb{G}_1^*$.

Extract, Encrypt. As in the FullIdent scheme. The only difference is that in Step 1 these algorithms compute $Q_{\mathsf{ID}} = L(H_1'(\mathsf{ID})) \in \mathbb{G}_1^*$.

Decrypt. As in the FullIdent scheme.

This completes the description of FullIdent'. The following theorem shows that FullIdent' is a chosen ciphertext secure IBE (i.e., IND-ID-CCA), assuming FullIdent is.

THEOREM 4.7. *Let $\mathcal{A}$ be an* IND-ID-CCA *adversary on* FullIdent' *that achieves advantage $\epsilon(k)$. Suppose $\mathcal{A}$ makes at most q_{H_1} queries to the hash function H_1'. Then there is an* IND-ID-CCA *adversary $\mathcal{B}$ on* FullIdent *that achieves the same advantage $\epsilon(k)$ and $time(\mathcal{B}) = time(\mathcal{A}) + q_{H_1} \cdot time(L_S)$*

Proof sketch. Algorithm $\mathcal{B}$ attacks FullIdent by running algorithm $\mathcal{A}$. It relays all decryption queries, extraction queries, and hash queries from $\mathcal{A}$ directly to the challenger and relays the challenger's response back to $\mathcal{A}$. It behaves differently only when $\mathcal{A}$ issues a hash query to H_1'. Recall that $\mathcal{B}$ has access only to a hash function $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$. To respond to H_1' queries, algorithm $\mathcal{B}$ maintains a list of tuples $\langle \mathsf{ID}_j, y_j \rangle$ as explained below. We refer to this list as the $(H_1')^{list}$. The list is initially empty. When $\mathcal{A}$ queries the oracle H_1' at a point ID_i, algorithm $\mathcal{B}$ responds as follows.

1. If the query ID_i already appears on the $(H_1')^{list}$ in a tuple $\langle \mathsf{ID}_i, y_i \rangle$, then respond with $H_1'(\mathsf{ID}_i) = y_i \in A$.
2. Otherwise, $\mathcal{B}$ issues a query for $H_1(\mathsf{ID}_i)$, say, $H_1(\mathsf{ID}_i) = \alpha \in \mathbb{G}_1^*$.
3. $\mathcal{B}$ runs the sampling algorithm $\mathcal{L}_S(\alpha)$ to generate a random element $y \in L^{-1}(\alpha)$.
4. $\mathcal{B}$ adds the tuple $\langle \mathsf{ID}_i, y \rangle$ to the $(H_1')^{list}$ and responds to $\mathcal{A}$ with $H_1'(\mathsf{ID}_i) = y \in A$. Note that y is uniformly distributed in A as required since α is uniformly distributed in $\mathbb{G}_1^*$ and L is an ℓ-to-1 map.

Algorithm $\mathcal{B}$'s responses to all of $\mathcal{A}$'s queries, including H_1' queries, are identical to $\mathcal{A}$'s view in the real attack. Hence $\mathcal{B}$ will have the same advantage $\epsilon(k)$ in winning the game with the challenger. □

5. A concrete IBE system using the Weil pairing. In this section, we use FullIdent' to describe a concrete IBE system based on the Weil pairing. We first review some properties of the pairing (see the appendix for more details).

5.1. Properties of the Weil pairing. Let $p > 3$ be a prime satisfying $p = 2 \bmod 3$, and let q be some prime factor of $p+1$. Let E be the elliptic curve defined by the equation $y^2 = x^3 + 1$ over $\mathbb{F}_p$. We state a few elementary facts about this curve E (see [43] for more information). From here on, we let $E(\mathbb{F}_{p^r})$ denote the group of

points on E defined over $\mathbb{F}_{p^r}$.

Fact 1. Since $x^3 + 1$ is a permutation on $\mathbb{F}_p$, it follows that the group $E(\mathbb{F}_p)$ contains $p + 1$ points. We let O denote the point at infinity. Let $P \in E(\mathbb{F}_p)$ be a point of order q, and let $\mathbb{G}_1$ be the subgroup of points generated by P.

Fact 2. For any $y_0 \in \mathbb{F}_p$, there is a unique point (x_0, y_0) on $E(\mathbb{F}_p)$, namely, $x_0 = (y_0^2 - 1)^{1/3} \in \mathbb{F}_p$. Hence, if (x, y) is a random nonzero point on $E(\mathbb{F}_p)$, then y is uniform in $\mathbb{F}_p$. We use this property to build a simple admissible encoding function.

Fact 3. Let $1 \neq \zeta \in \mathbb{F}_{p^2}$ be a solution of $x^3 - 1 = 0 \bmod p$. Then the map $\phi(x, y) = (\zeta x, y)$ is an automorphism of the group of points on the curve E. Note that, for any point $Q = (x, y) \in E(\mathbb{F}_p)$, we have that $\phi(Q) \in E(\mathbb{F}_{p^2})$, but $\phi(Q) \notin E(\mathbb{F}_p)$. Hence $Q \in E(\mathbb{F}_p)$ is linearly independent of $\phi(Q) \in E(\mathbb{F}_{p^2})$.

Fact 4. Since the points $P \in \mathbb{G}_1$ and $\phi(P)$ are linearly independent, they generate a group isomorphic to $\mathbb{Z}_q \times \mathbb{Z}_q$. We denote this group of points by $E[q]$.

Let $\mathbb{G}_2$ be the subgroup of $\mathbb{F}_{p^2}^*$ of order q. The Weil pairing on the curve $E(\mathbb{F}_{p^2})$ is a mapping $e : E[q] \times E[q] \to \mathbb{G}_2$ defined in the appendix. For any $Q, R \in E(\mathbb{F}_p)$, the Weil pairing satisfies $e(Q, R) = 1$. In other words, the Weil pairing is degenerate on $E(\mathbb{F}_p)$ and hence degenerate on the group $\mathbb{G}_1$. To get a nondegenerate map, we define the modified Weil pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ as follows:

$$\hat{e}(P, Q) = e(P, \phi(Q)).$$

The modified Weil pairing satisfies the following properties:

1. *Bilinear.* For all $P, Q \in \mathbb{G}_1$ and for all $a, b \in \mathbb{Z}$, we have $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$.
2. *Nondegenerate.* If P is a generator of $\mathbb{G}_1$, then $\hat{e}(P, P) \in \mathbb{F}_{p^2}^*$ is a generator of $\mathbb{G}_2$.
3. *Computable.* Given $P, Q \in \mathbb{G}_1$, there is an efficient algorithm, due to Miller, to compute $\hat{e}(P, Q) \in \mathbb{G}_2$. This algorithm is described in the appendix. Its running time is comparable to exponentiation in $\mathbb{F}_p$.

Joux and Nguyen [28] point out that, although the CDH problem appears to be hard in the group $\mathbb{G}_1$, the DDH problem is easy in $\mathbb{G}_1$ (as discussed in section 3).

BDH parameter generator $\mathcal{G}_1$. Given a security parameter $2 < k \in \mathbb{Z}$, the BDH parameter generator picks a random k-bit prime q and finds the smallest prime p such that (1) $p = 2 \bmod 3$, (2) q divides $p + 1$, and (3) q^2 does not divide $p + 1$. We write $p = \ell q + 1$. The group $\mathbb{G}_1$ is the subgroup of order q of the group of points on the curve $y^2 = x^3 + 1$ over $\mathbb{F}_p$. The group $\mathbb{G}_2$ is the subgroup of order q of $\mathbb{F}_{p^2}^*$. The bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is the modified Weil pairing defined above.

The BDH parameter generator $\mathcal{G}_1$ is believed to satisfy the BDH assumption asymptotically. However, there is still the question of what values of p and q can be used in practice to make the BDH problem sufficiently hard. At the very least, we must ensure that the discrete log problem in $\mathbb{G}_1$ is sufficiently hard. As pointed out in section 3, the discrete log problem in $\mathbb{G}_1$ is efficiently reducible to discrete log in $\mathbb{G}_2$ (see [32, 17]). Hence computing discrete log in $\mathbb{F}_{p^2}^*$ is sufficient for computing discrete log in $\mathbb{G}_1$. In practice, for proper security of discrete log in $\mathbb{F}_{p^2}^*$, one often uses primes p that are at least 512 bits long (so that the group size is at least 1024 bits long). Consequently, one should not use this BDH parameter generator with primes p that are less than 512 bits long.

5.2. An admissible encoding function: MapToPoint. Let $\mathbb{G}_1, \mathbb{G}_2$ be two groups generated by $\mathcal{G}_1$ as defined above. Recall that the IBE system of section 4.2 uses a hash function $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$. By Theorem 4.7, it suffices to have a

hash function $H_1 : \{0,1\}^* \to A$ for some set A and an admissible encoding function $L : A \to \mathbb{G}_1^*$. In what follows, the set A will be $\mathbb{F}_p$, and the admissible encoding function L will be called MapToPoint.

Let p be a prime satisfying $p = 2 \bmod 3$ and $p = \ell q - 1$ for some prime $q > 3$. We require that q does not divide ℓ (i.e., that q^2 does not divide $p+1$). Let E be the elliptic curve $y^2 = x^3 + 1$ over $\mathbb{F}_p$. Let $\mathbb{G}_1$ be the subgroup of points on E of order q. Suppose we already have a hash function $H_1 : \{0,1\}^* \to \mathbb{F}_p$.

Algorithm MapToPoint works as follows on input $y_0 \in \mathbb{F}_p$:

1. Compute $x_0 = (y_0^2 - 1)^{1/3} = (y_0^2 - 1)^{(2p-1)/3} \in \mathbb{F}_p$.
2. Let $Q = (x_0, y_0) \in E(\mathbb{F}_p)$, and set $Q_{\mathsf{ID}} = \ell Q \in \mathbb{G}_1$.
3. Output $\mathsf{MapToPoint}(y_0) = Q_{\mathsf{ID}}$.

This completes the description of MapToPoint.

We note that there are $\ell - 1$ values of $y_0 \in \mathbb{F}_p$ for which $\ell Q = \ell(x_0, y_0) = O$. (These are the non-$O$ points of order dividing ℓ.) Let $B \subset \mathbb{F}_p$ be the set of these y_0. When $H_1(\mathsf{ID})$ is one of these $\ell - 1$ values, Q_{ID} is the identity element of $\mathbb{G}_1$. It is extremely unlikely for $H_1(\mathsf{ID})$ to hit one of these points—the probability is $1/q < 1/2^k$. Hence, for simplicity, we say that $H_1(\mathsf{ID})$ outputs elements only in $\mathbb{F}_p \setminus B$, i.e., $H_1 : \{0,1\}^* \to \mathbb{F}_p \setminus B$. Algorithm MapToPoint can be easily extended to handle the values $y_0 \in B$ by hashing ID multiple times using different hash functions.

LEMMA 5.1. *$\mathsf{MapToPoint} : \mathbb{F}_p \setminus B \to \mathbb{G}_1^*$ is an admissible encoding function.*

Proof. The map is clearly computable and is an ℓ-to-1 mapping. It remains to show that L is samplable. Let P be a generator of $E(\mathbb{F}_p)$. Given a $Q \in \mathbb{G}_1^*$, the sampling algorithm $\mathcal{L}_S$ does the following: (1) pick a random $b \in \{0, \ldots, \ell - 1\}$, (2) compute $Q' = \ell^{-1} \cdot Q + bqP = (x, y)$, and (3) output $\mathcal{L}_S(Q) = y \in \mathbb{F}_p$. Here ℓ^{-1} is the inverse of ℓ in $\mathbb{Z}_q^*$. This algorithm outputs a random element from the ℓ elements in $\mathsf{MapToPoint}^{-1}(Q)$ as required. ∎

5.3. A concrete IBE system. Using FullIdent' from section 4.3 with the BDH parameter generator $\mathcal{G}_1$ and the admissible encoding function MapToPoint, we obtain a concrete IBE system. Note that, in this system, H_1 is a hash function from $\{0,1\}^*$ to $\mathbb{F}_p$ (where p is the finite field output by $\mathcal{G}_1$). The security of the system follows directly from Theorems 4.4 and 4.7. We summarize this in the following corollary.

COROLLARY 5.2. *The IBE system* FullIdent' *using the BDH parameter generator $\mathcal{G}_1$ and the admissible encoding* MapToPoint *is a chosen ciphertext secure IBE (i.e.,* IND-ID-CCA *in the random oracle model) assuming $\mathcal{G}_1$ satisfies the BDH assumption.*

Performance. Algorithms Setup and Extract are very simple. At the heart of both algorithms is a standard multiplication on the curve $E(\mathbb{F}_p)$. Algorithm Encrypt requires that the encryptor compute the Weil pairing of Q_{ID} and P_{pub}. Note that this computation is independent of the message to be encrypted and hence can be done once and for all. Once g_{ID} is computed, the performance of the system is almost identical to standard ElGamal encryption. Decryption is a single Weil pairing computation. We note that the ciphertext length of BasicIdent using $\mathcal{G}_1$ is the same as in regular ElGamal encryption in $\mathbb{F}_p$.

6. Extensions and observations. We briefly describe a few extensions to the IBE scheme of the previous sections.

Tate pairing and other curves. Our IBE system works with any efficiently computable bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ between two groups $\mathbb{G}_1, \mathbb{G}_2$ as long as the BDH assumption holds. Many different curves, or more generally Abelian varieties, are believed to give rise to such maps. For example, one could use the curve

$y^2 = x^3 + x$ over $\mathbb{F}_p$ with $p = 3 \bmod 4$ and its endomorphism $\phi : (x, y) \to (-x, iy)$, where $i^2 = -1$. As another example, Galbraith [18] suggests using supersingular elliptic curves over a field of small characteristic to reduce the ciphertext size in our system. More general Abelian varieties are proposed by Rubin and Silverberg [39]. We note that both encryption and decryption in FullIdent can be made faster by using the Tate pairing on elliptic curves rather than the Weil pairing [19, 1].

Asymmetric pairings. Our IBE system can use slightly more general bilinear maps, namely, maps of the form $\hat{e} : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_2$, where $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_2$ are three groups of prime order q. Using the notation of section 4.1, the only change to BasicIdent is that we take P and P_{pub} as elements in $\mathbb{G}_0$ and let H_1 be a hash function $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$. Everything else remains the same. However, to make the proof of security go through (Lemma 4.2 in particular), we need a different complexity assumption which we call the co-BDH assumption: given random $P, aP, bP \in \mathbb{G}_0$ and $Q, aQ, cQ \in \mathbb{G}_1$, no polynomial time algorithm can compute $\hat{e}(P, Q)^{abc}$ with nonnegligible probability. If one is willing to accept this assumption, then we can avoid using supersingular curves and instead use elliptic curves over $\mathbb{F}_p$, $p > 3$, proposed by Miyaji, Nakabayashi, and Takano [35]. Curves $E/\mathbb{F}_p$ in this family are not supersingular and have the property that if q divides $|E(F_p)|$, then $E[q] \subseteq E(F_{p^6})$. (Recall that $E[q]$ is the group containing all points in E of order dividing q.) One way to use these curves is to set $\mathbb{G}_1$ to be a cyclic subgroup of $E(\mathbb{F}_p)$ of order q and $\mathbb{G}_0$ to be a different cyclic subgroup of $E(\mathbb{F}_{p^6})$ of the same order q. The standard Weil or Tate pairings on $\mathbb{G}_0 \times \mathbb{G}_1$ can be used as the bilinear map $\hat{e}$. Note that hashing public keys onto $\mathbb{G}_1 \subseteq E(\mathbb{F}_p)$ is easily done. Alternatively, to reduce the ciphertext size (which contains an element from $\mathbb{G}_0$), one could take $\mathbb{G}_0$ as a subgroup of order q of $E(\mathbb{F}_p)$ and $\mathbb{G}_1$ as a different subgroup of $E(\mathbb{F}_{p^6})$ of the same order. The question is how to hash public keys into $\mathbb{G}_1$. To do so, let $\mathrm{tr} : E(\mathbb{F}_{p^6}) \to E(\mathbb{F}_p)$ be the trace map on the curve, and define $\mathbb{G}_1$ to be the subgroup of $E[q]$ containing all points P whose trace is O, i.e., $\mathrm{tr}(P) = O$. Then, given a hash function $H : \{0,1\}^* \to E[q]$, we can hash a public key ID into $\mathbb{G}_1$ by computing $H_1(\mathsf{ID}) = 6H(\mathsf{ID}) - \mathrm{tr}(H(\mathsf{ID})) \in \mathbb{G}_1$. Finally, we note that by modifying the security proof appropriately, one can take $\mathbb{G}_1 = E[q]$ (a noncyclic group) and then avoid computing traces while hashing into $\mathbb{G}_1$ (see also [18]).

Distributed PKG. In the standard use of an IBE in an e-mail system, the master-key stored at the PKG must be protected in the same way that the private key of a CA is protected. One way of protecting this key is by distributing it among different sites using techniques of threshold cryptography [20]. Our IBE system supports this in a very efficient and robust way. Recall that the master-key is some $s \in \mathbb{Z}_q^*$. In order to generate a private key, the PKG computes $Q_{priv} = sQ_{\mathsf{ID}}$, where Q_{ID} is derived from the user's public key ID. This can easily be distributed in a t-out-of-n fashion by giving each of the n PKGs one share s_i of a Shamir secret sharing of $s \bmod q$. When generating a private key, each of the t chosen PKGs simply responds with $Q_{priv}^{(i)} = s_i Q_{\mathsf{ID}}$. The user can then construct Q_{priv} as $Q_{priv} = \sum \lambda_i Q_{priv}^{(i)}$, where the λ_i's are the appropriate Lagrange coefficients.

Furthermore, it is easy to make this scheme robust against dishonest PKGs using the fact that DDH is easy in $\mathbb{G}_1$. During the set-up phase, each of the n PKGs publishes $P_{pub}^{(i)} = s_i P$. During a key generation request, the user can verify that the response from the ith PKG is valid by testing that

$$\hat{e}(Q_{priv}^{(i)}, P) = \hat{e}(Q_{\mathsf{ID}}, P_{pub}^{(i)}).$$

Thus a misbehaving PKG will be caught immediately. There is no need for zero-

knowledge proofs as in regular robust threshold schemes [21]. The PKG's master-key can be generated in a distributed fashion using the techniques of [22].

Note that a distributed master-key also enables threshold decryption on a *per-message* basis, without any need to derive the corresponding decryption key. For example, threshold decryption of BasicIdent ciphertext (U, V) is straightforward if each PKG responds with $\hat{e}(s_i Q_{\mathsf{ID}}, U)$.

Working in subgroups. The performance of our IBE system (section 5) can be improved if we work in a small subgroup of the curve. For example, choose a 1024-bit prime $p = 2 \bmod 3$ with $p = aq - 1$ for some 160-bit prime q. The point P is then chosen to be a point of order q. Each public key ID is converted to a group point by hashing ID to a point Q on the curve and then multiplying the point by a. The system is secure if the BDH assumption holds in the group generated by P. The advantage is that the Weil computation is done on points of small order and hence is much faster.

IBE implies signatures. Moni Naor has observed that an IBE scheme can be immediately converted into a public key signature scheme. The reasoning is as follows. The private key for the signature scheme is the master key for the IBE scheme. The public key for the signature scheme is the global system parameters for the IBE scheme. The signature on a message M is the IBE decryption key for $\mathsf{ID} = M$. To verify a signature, choose a random message M', encrypt M' using the public key $\mathsf{ID} = M$, and then attempt to decrypt using the given signature on M as the decryption key. If the IBE scheme is IND-ID-CCA, then the signature scheme is existentially unforgeable against a chosen message attack. Note that, unlike most signature schemes, the signature verification algorithm here is randomized. This shows that secure IBE schemes incorporate both public key encryption and digital signatures. We note that the signature scheme derived from our IBE system has some interesting properties [6].

7. Escrow ElGamal encryption. In this section, we show that the Weil pairing enables us to add a global escrow capability to the ElGamal encryption system. A single escrow key enables the decryption of ciphertexts encrypted under any public key. Paillier and Yung have shown how to add a global escrow capability to the Paillier encryption system [36]. Our ElGamal escrow system works as follows.

Setup. Let $\mathcal{G}$ be some BDH parameter generator. Given a security parameter $k \in \mathbb{Z}^+$, the algorithm works as follows.

Step 1: Run $\mathcal{G}$ on input k to generate a prime q, two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, and an admissible bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Let P be a generator of $\mathbb{G}_1$.

Step 2: Pick a random $s \in \mathbb{Z}_q^*$, and set $Q = sP$.

Step 3: Choose a cryptographic hash function $H : \mathbb{G}_2 \to \{0,1\}^n$.

The message space is $\mathcal{M} = \{0,1\}^n$. The ciphertext space is $\mathcal{C} = \mathbb{G}_1 \times \{0,1\}^n$. The system parameters are $\mathsf{params} = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, Q, H \rangle$. The escrow key is $s \in \mathbb{Z}_q^*$.

keygen. A user generates a public/private key pair for herself by picking a random $x \in \mathbb{Z}_q^*$ and computing $P_{pub} = xP \in \mathbb{G}_1$. Her private key is x, and her public key is P_{pub}.

Encrypt. To encrypt $M \in \{0,1\}^n$ under the public key P_{pub}, do the following: (1) pick a random $r \in \mathbb{Z}_q^*$, and (2) set the ciphertext to be

$$C = \langle rP,\ M \oplus H(g^r) \rangle, \quad \text{where} \quad g = \hat{e}(P_{pub}, Q) \in \mathbb{G}_2.$$

Decrypt. Let $C = \langle U, V \rangle$ be a ciphertext encrypted using P_{pub}. Then $U \in \mathbb{G}_1$. To decrypt C using the private key x, do:

$$V \oplus H(\hat{e}(U, xQ)) = M.$$

Escrow-decrypt. To decrypt $C = \langle U, V \rangle$ using the escrow key s, do:

$$V \oplus H(\hat{e}(U, sP_{pub})) = M.$$

A standard argument shows that assuming that BDH is hard for groups generated by $\mathcal{G}$ the system has semantic security in the random oracle model. (Recall that, since DDH is easy, we cannot prove semantic security based on DDH.) Yet the escrow agent can decrypt any ciphertext encrypted using any user's public key. The decryption capability of the escrow agent can be distributed using the PKG distribution techniques described in section 6.

Using a similar hardness assumption, Verheul [46] described an ElGamal encryption system with nonglobal escrow. Each user constructs a public key with two corresponding private keys and gives one of the private keys to the trusted third party. The trusted third party must maintain a database of all private keys given to it by the various users.

8. Summary and open problems. We defined chosen ciphertext security for identity-based systems and proposed a fully functional IBE system. The system has chosen ciphertext security in the random oracle model assuming BDH, a natural analogue of the CDH problem. The BDH assumption deserves further study considering the powerful cryptosystems derived from it. For example, it could be interesting to see whether the techniques of [30] can be used to prove that the BDH assumption is equivalent to the discrete log assumption on the curve for certain primes p.

Recently, Cocks [8] proposed another IBE system whose security is based on the difficulty of distinguishing quadratic residues from nonresidues in the ring $\mathbb{Z}/N\mathbb{Z}$, where N is an RSA modulus (i.e., a product of two large primes). Cocks' system is somewhat harder to use in practice than the IBE system in this paper. Cocks' system uses bit-by-bit encryption and consequently outputs long ciphertexts. Also, encryption/decryption is a bit slower than the system described in this paper. Nevertheless, it is encouraging to see that IBE systems can be built using very different complexity assumptions.

It is an open problem to build chosen ciphertext secure identity-based systems that are secure in the standard computation model (rather than the random oracle model). One might hope to use the techniques of Cramer–Shoup [10] to provide chosen ciphertext security based on DDH. Unfortunately, as mentioned in section 3, the DDH assumption is false in the group of points on the curve E. However, simple variants of DDH do seem to hold. In particular, the following two distributions appear to be computationally indistinguishable: $\langle P, aP, bP, cP, abcP \rangle$ and $\langle P, aP, bP, cP, rP \rangle$, where a, b, c, r are random in $\mathbb{Z}_q$. We refer to this assumption as BDDH. A chosen ciphertext secure identity-based system strictly based on BDDH would be a plausible analogue of the Cramer–Shoup system. Building a chosen ciphertext secure IBE (IND-ID-CCA) in the standard model is currently an open problem.

Appendix A. Definition of the Weil pairing. We define the Weil pairing and show how to efficiently compute it using an algorithm due to Miller [34]. To be concrete, we present the algorithm as it applies to supersingular elliptic curves defined

over a prime field $\mathbb{F}_p$ with $p > 3$. (The curve $y^2 = x^3 + 1$ over $\mathbb{F}_p$ with $p = 2 \bmod 3$ is an example of such a curve.) The definition and algorithm easily generalize to computing the Weil pairing over other elliptic curves. We state a few elementary facts about such curves [43].

Fact 1. A supersingular curve $E/\mathbb{F}_p$ (with $p > 3$) contains $p+1$ points in $\mathbb{F}_p$. We let O denote the point at infinity. The group of points over $\mathbb{F}_p$ forms a cyclic group of order $p+1$. Let $P \in E(\mathbb{F}_p)$ be a point order n, where n divides $p+1$.

Fact 2. The group of points $E(\mathbb{F}_{p^2})$ contains a point Q of order n which is linearly independent of the points in $E(\mathbb{F}_p)$. Hence $E(\mathbb{F}_{p^2})$ contains a subgroup which is isomorphic to the group $\mathbb{Z}_n^2$. The group is generated by $P \in E(\mathbb{F}_p)$ and $Q \in E(\mathbb{F}_{p^2})$. We denote this group by $E[n]$.

Throughout this section, we let $\mathbb{G}_2$ denote the subgroup of $\mathbb{F}_{p^2}^*$ of order n. We will be working with the Weil pairing e which maps pairs of points in $E[n]$ to $\mathbb{G}_2$, i.e., $e : E[n] \times E[n] \to \mathbb{G}_2$. To define the pairing, we review a few basic concepts (see [29, pp. 243–245]). In what follows, we let P and Q be arbitrary points in $E(\mathbb{F}_{p^2})$.

Divisors. A divisor is a formal sum of points on the curve $E(\mathbb{F}_{p^2})$. We write divisors as $\mathcal{A} = \sum_P a_p(P)$, where $a_P \in \mathbb{Z}$ and $P \in E(\mathbb{F}_{p^2})$. For example, $\mathcal{A} = 3(P_1) - 2(P_2) - (P_3)$ is a divisor. We will consider only divisors $\mathcal{A} = \sum_P a_p(P)$, where $\sum_P a_p = 0$.

Functions. Roughly speaking, a function f on the curve $E(\mathbb{F}_{p^2})$ can be viewed as a rational function $f(x, y) \in \mathbb{F}_{p^2}(x, y)$. For any point $P = (x, y) \in E(\mathbb{F}_{p^2})$, we define $f(P) = f(x, y)$.

Divisors of functions. Let f be a function on the curve $E(\mathbb{F}_{p^2})$. We define its divisor, denoted by (f), as $(f) = \sum_P \mathrm{ord}_P(f) \cdot (P)$. Here $\mathrm{ord}_P(f)$ is the order of the zero that f has at the point P. For example, let $ax + by + c = 0$ be the line passing through the points $P_1, P_2 \in E(\mathbb{F}_{p^2})$ with $P_1 \neq \pm P_2$. This line intersects the curve at a third point $P_3 \in E(\mathbb{F}_{p^2})$. Then the function $f(x, y) = ax + by + c$ has three zeroes P_1, P_2, P_3 and a pole of order 3 at infinity. The divisor of f is $(f) = (P_1) + (P_2) + (P_3) - 3(O)$.

Principal divisors. Let $\mathcal{A}$ be a divisor. If there exists a function f such that $(f) = \mathcal{A}$, then we say that $\mathcal{A}$ is a principal divisor. We know that a divisor $\mathcal{A} = \sum_P a_p(P)$ is principal if and only if $\sum_P a_p = 0$ and $\sum_P a_P P = O$. Note that the second summation is using the group action on the curve. Furthermore, given a principal divisor $\mathcal{A}$, there exists a *unique* function f (up to constant multiples) such that $(A) = (f)$.

Equivalence of divisors. We say that two divisors $\mathcal{A}, \mathcal{B}$ are equivalent if their difference $\mathcal{A} - \mathcal{B}$ is a principal divisor. We know that any divisor $\mathcal{A} = \sum_P a_p(P)$ (with $\sum_P a_P = 0$) is equivalent to a divisor of the form $\mathcal{A}' = (Q) - (O)$ for some $Q \in E$. Observe that $Q = \sum_P a_P P$.

Notation. Given a function f and a divisor $\mathcal{A} = \sum_P a_p(P)$, we define $f(\mathcal{A})$ as $f(\mathcal{A}) = \prod_P f(P)^{a_P}$. Note that, since $\sum_P a_P = 0$, we have that $f(\mathcal{A})$ remains unchanged if instead of f we use cf for any $c \in \mathbb{F}_{p^2}$.

We are now ready to define the Weil pairing of two points $P, Q \in E[n]$. Let $\mathcal{A}_P$ be some divisor equivalent to the divisor $(P) - (O)$. We know that $n\mathcal{A}_P$ is a principal divisor (it is equivalent to $n(P) - n(O)$ which is clearly a principal divisor). Hence there exists a function f_P such that $(f_P) = n\mathcal{A}_P$. Define $\mathcal{A}_Q$ and f_Q analogously. The Weil pairing of P and Q is defined as

$$e(P, Q) = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)}.$$

This ratio defines the Weil pairing of P and Q whenever it is well defined (no division by zero occurred). If this ratio is undefined, we use different divisors $\mathcal{A}_P, \mathcal{A}_Q$ to define $e(P,Q)$.

We briefly show that the Weil pairing is well defined. That is, the value of $e(P,Q)$ is independent of the choice of the divisor $\mathcal{A}_P$ as long as $\mathcal{A}_P$ is equivalent to $(P)-(O)$ and $\mathcal{A}_P$ leads to a well-defined value. The same holds for $\mathcal{A}_Q$. Let $\hat{\mathcal{A}}_P$ be a divisor equivalent to $\mathcal{A}_P$, and let $\hat{f}_P$ be a function so that $(\hat{f}_P) = n\hat{\mathcal{A}}_P$. Then $\hat{\mathcal{A}}_P = \mathcal{A}_P + (g)$ for some function g and $\hat{f}_P = f_P \cdot g^n$. We have that

$$\begin{aligned} e(P,Q) &= \frac{\hat{f}_P(\mathcal{A}_Q)}{f_Q(\hat{\mathcal{A}}_P)} = \frac{f_P(\mathcal{A}_Q)g(\mathcal{A}_Q)^n}{f_Q(\mathcal{A}_P)f_Q((g))} = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)} \cdot \frac{g(n\mathcal{A}_Q)}{f_Q((g))} \\ &= \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)} \cdot \frac{g((f_Q))}{f_Q((g))} = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)}. \end{aligned}$$

The last equality follows from the following fact known as Weil reciprocity: for any two functions f, g, we have that $f((g)) = g((f))$. Hence the Weil pairing is well defined.

FACT A.1. *The Weil pairing has the following properties for points in $E[n]$:*

- *For all $P \in E[n]$, we have $e(P,P) = 1$.*
- *Bilinear: $e(P_1 + P_2, Q) = e(P_1, Q) \cdot e(P_2, Q)$ and $e(P, Q_1 + Q_2) = e(P, Q_1) \cdot e(P, Q_2)$.*
- *When $P, Q \in E[n]$ are collinear, $e(P,Q) = 1$. Similarly, $e(P,Q) = e(Q,P)^{-1}$.*
- *nth root: for all $P, Q \in E[n]$, we have $e(P,Q)^n = 1$, i.e., $e(P,Q) \in \mathbb{G}_2$.*
- *Nondegenerate in the following sense: if $P \in E[n]$ satisfies $e(P,Q) = 1$ for all $Q \in E[n]$, then $P = O$.*

As discussed in section 5, our concrete IBE scheme uses the modified Weil pairing $\hat{e}(P,Q) = e(P, \phi(Q))$, where ϕ is an automorphism on the group of points of E.

Tate pairing. The Tate pairing [17] is another bilinear pairing that has the required properties for our system. We slightly modify the original definition to fit our purpose. Define the Tate pairing of two points $P, Q \in E[n]$ as $T(P,Q) = f_P(\mathcal{A}_Q)^{|\mathbb{F}_{p^2}^*|/n}$, where f_P and $\mathcal{A}_Q$ are defined as above. This definition gives a computable bilinear pairing $T : E[n] \times E[n] \to \mathbb{G}_2$.

Appendix B. Computing the Weil pairing. Given two points $P, Q \in E[n]$, we show how to compute $e(P,Q) \in \mathbb{F}_{p^2}^*$ using $O(\log p)$ arithmetic operations in $\mathbb{F}_p$. We assume $P \neq Q$. We proceed as follows: pick two random points $R_1, R_2 \in E[n]$. Consider the divisors $\mathcal{A}_P = (P + R_1) - (R_1)$ and $\mathcal{A}_Q = (Q + R_2) - (R_2)$. These divisors are equivalent to $(P) - (O)$ and $(Q) - (O)$, respectively. Hence we can use $\mathcal{A}_P$ and $\mathcal{A}_Q$ to compute the Weil pairing as

$$e(P,Q) = \frac{f_P(\mathcal{A}_Q)}{f_Q(\mathcal{A}_P)} = \frac{f_P(Q + R_2) f_Q(R_1)}{f_P(R_2) f_Q(P + R_1)}.$$

This expression is well defined with very high probability over the choice of R_1, R_2. (The probability of failure is at most $O(\frac{\log p}{p})$.) In the rare event that a division by zero occurs during the computation of $e(P,Q)$, we simply pick new random points R_1, R_2 and repeat the process.

To evaluate $e(P,Q)$, it suffices to show how to evaluate the function f_P at $\mathcal{A}_Q$. Evaluating $f_Q(\mathcal{A}_P)$ is done analogously. We evaluate $f_P(\mathcal{A}_Q)$ using repeated doubling. For a positive integer b, define the divisor

$$\mathcal{A}_b = b(P + R_1) - b(R_1) - (bP) + (O).$$

It is a principal divisor, and therefore there exists a function f_b such that $(f_b) = \mathcal{A}_b$. Observe that $(f_P) = (f_n)$ and hence $f_P(\mathcal{A}_Q) = f_n(\mathcal{A}_Q)$. It suffices to show how to evaluate $f_n(\mathcal{A}_Q)$.

LEMMA B.1. *There is an algorithm $\mathcal{D}$ that given $f_b(\mathcal{A}_Q), f_c(\mathcal{A}_Q)$ and $bP, cP, (b+c)P$ for some $b, c > 0$ outputs $f_{b+c}(\mathcal{A}_Q)$. The algorithm uses only a (small) constant number of arithmetic operations in $\mathbb{F}_{p^2}$.*

Proof. We first define two auxiliary linear functions g_1, g_2:

1. Let $a_1x + b_1y + c_1 = 0$ be the line passing through the points bP and cP. (If $b = c$, then let $a_1x + b_1y + c_1 = 0$ be the line tangent to E at bP.) Define $g_1(x, y) = a_1x + b_1y + c_1$.
2. Let $x + c_2 = 0$ be the vertical line passing through the point $(b+c)P$. Define $g_2(x, y) = x + c_2$.

The divisors of these functions are

$$(g_1) = (bP) + (cP) + (-(b+c)P) - 3(O),$$
$$(g_2) = ((b+c)P) + (-(b+c)P) - 2(O).$$

By definition we have that

$$\mathcal{A}_b = b(P + R_1) - b(R_1) - (bP) + (O),$$
$$\mathcal{A}_c = c(P + R_1) - c(R_1) - (cP) + (O),$$
$$\mathcal{A}_{b+c} = (b+c)(P + R_1) - (b+c)(R_1) - ((b+c)P) + (O).$$

It now follows that $\mathcal{A}_{b+c} = \mathcal{A}_b + \mathcal{A}_c + (g_1) - (g_2)$. Hence

$$f_{b+c}(\mathcal{A}_Q) = f_b(\mathcal{A}_Q) \cdot f_c(\mathcal{A}_Q) \cdot \frac{g_1(\mathcal{A}_Q)}{g_2(\mathcal{A}_Q)}. \tag{B.1}$$

This shows that to evaluate $f_{b+c}(\mathcal{A}_Q)$ it suffices to evaluate $g_i(\mathcal{A}_Q)$ for all $i = 1, 2$ and plug the results into (B.1). Hence, given $f_b(\mathcal{A}_Q), f_c(\mathcal{A}_Q)$ and $bP, cP, (b+c)P$, one can compute $f_{b+c}(\mathcal{A}_Q)$ using a constant number of arithmetic operations. ∎

We let $\mathcal{D}\big(f_b(\mathcal{A}_Q), f_c(\mathcal{A}_Q), bP, cP, (b+c)P\big) = f_{b+c}(\mathcal{A}_Q)$ denote the output of Algorithm $\mathcal{D}$ defined in Lemma B.1. Then one can compute $f_P(\mathcal{A}_Q) = f_n(\mathcal{A}_Q)$ using the following standard repeated doubling procedure. Let $n = b_m b_{m-1} \dots b_1 b_0$ be the binary representation of n, i.e., $n = \sum_{i=0}^{m} b_i 2^i$.

Init: Set $Z = O$, $V = f_0(\mathcal{A}_Q) = 1$, and $k = 0$.

Iterate: For $i = m, m-1, \dots, 1, 0$ do:

1: If $b_i = 1$, then do: Set $V = \mathcal{D}(V, f_1(\mathcal{A}_Q), Z, P, Z + P)$, set $Z = Z + P$, and set $k = k + 1$.

2: If $i > 0$, set $V = \mathcal{D}(V, V, Z, Z, 2Z)$, set $Z = 2Z$, and set $k = 2k$.

3: Observe that at the end of each iteration we have $Z = kP$ and $V = f_k(\mathcal{A}_Q)$.

Output: After the last iteration, we have $k = n$, and therefore $V = f_n(\mathcal{A}_Q)$ as required.

To evaluate the Weil pairing $e(P, Q)$, we run the above algorithm once to compute $f_P(\mathcal{A}_Q)$ and once to compute $f_Q(\mathcal{A}_P)$. The Tate pairing is evaluated similarly. Note that the repeated squaring algorithm needs to evaluate $f_1(\mathcal{A}_Q)$. This is easily done since the function $f_1(x, y)$ (whose divisor is $(f_1) = (P + R_1) - (R_1) - (P) + (O)$) can be written out explicitly as follows.

1. Let $a_1x + b_1y + c_1 = 0$ be the line passing through the points P and R_1. Define the function $g_1(x, y) = a_1x + b_1y + c_1$.

2. Let $x + c_2 = 0$ be the vertical line passing through the point $P + R_1$. Define the function $g_2(x, y) = x + c_2$.
3. The function $f_1(x, y)$ is simply $f_1(x, y) = g_2(x, y)/g_1(x, y)$, which is easy to evaluate in $\mathbb{F}_{p^2}$.

Acknowledgments. The authors thank Moni Naor, Alice Silverberg, Ben Lynn, Steven Galbraith, Kenny Paterson, and Mike Scott for helpful discussions about this work.

REFERENCES

[1] P. Barreto, H. Kim, B. Lynn, and M. Scott, *Efficient algorithms for pairing-based cryptosystems*, in Advances in Cryptology—Crypto 2002, Lecture Notes in Comput. Sci. 2140, Springer-Verlag, Berlin, 2002, pp. 354–368.

[2] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, *Relations among notions of security for public-key encryption schemes*, in Advances in Cryptology—Crypto 1998, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, Berlin, 1998, pp. 26–45.

[3] M. Bellare and P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, in ACM Conference on Computers and Communication Security, ACM, New York, 1993, pp. 62–73.

[4] D. Boneh, *The decision Diffie-Hellman problem*, in Proceedings of the Third Algorithmic Number Theory Symposium, Lecture Notes in Comput. Sci. 1423, Springer-Verlag, Berlin, 1998, pp. 48–63.

[5] D. Boneh and M. Franklin, *Identity-based encryption from the Weil pairing*, extended abstract in Advances in Cryptology—Crypto 2001, Lecture Notes in Comput. Sci. 2139, Springer-Verlag, Berlin, 2001, pp. 231–229; also available online from http://eprint.iacr.org/2001/090/.

[6] D. Boneh, B. Lynn, and H. Shacham, *Short signatures from the Weil pairing*, in Advances in Cryptology—AsiaCrypt 2001, Lecture Notes in Comput. Sci. 2248, Springer-Verlag, Berlin, 2001, pp. 514–532.

[7] M. Bellare, A. Boldyreva, and S. Micali, *Public-key encryption in a multi-user setting: Security proofs and improvements*, in Advances in Cryptology—Eurocrypt 2000, Lecture Notes in Comput. Sci. 1807, Springer-Verlag, Berlin, 2000, pp. 259–274.

[8] C. Cocks, *An identity based encryption scheme based on quadratic residues*, in Proceedings of the Eighth IMA International Conference on Cryptography and Coding, Royal Agricultural College, Cirencester, UK, 2001, pp. 360–363.

[9] J. Coron, *On the exact security of Full-Domain-Hash*, in Advances in Cryptology—Crypto 2000, Lecture Notes in Comput. Sci. 1880, Springer-Verlag, Berlin, 2000, pp. 229–235.

[10] R. Cramer and V. Shoup, *A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack*, in Advances in Cryptology—Crypto 1998, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, Berlin, 1998, pp. 13–25.

[11] Y. Desmedt and J. Quisquater, *Public-key systems based on the difficulty of tampering*, in Advances in Cryptology—Crypto 1986, Lecture Notes in Comput. Sci. 263, Springer-Verlag, Berlin, 1986, pp. 111–117.

[12] G. Di Crescenzo, R. Ostrovsky, and S. Rajagopalan, *Conditional oblivious transfer and timed-release encryption*, in Advances in Cryptology—Eurocrypt 1999, Lecture Notes in Comput. Sci. 1592, Springer-Verlag, Berlin, 1999, pp. 74–89.

[13] D. Dolev, C. Dwork, and M. Naor, *Nonmalleable cryptography*, SIAM J. Comput., 30 (2000), pp. 391–437.

[14] U. Feige, A. Fiat, and A. Shamir, *Zero-knowledge proofs of identity*, J. Cryptology, 1 (1988), pp. 77–94.

[15] A. Fiat and A. Shamir, *How to prove yourself: Practical solutions to identification and signature problems*, in Advances in Cryptology—Crypto 1986, Lecture Notes in Comput. Sci. 263, Springer-Verlag, Berlin, 1986, pp. 186–194.

[16] E. Fujisaki and T. Okamoto, *Secure integration of asymmetric and symmetric encryption schemes*, in Advances in Cryptology—Crypto 1999, Lecture Notes in Comput. Sci. 1666, Springer-Verlag, Berlin, 1999, pp. 537–554.

[17] G. Frey, M. Müller, and H. Rück, *The Tate pairing and the discrete logarithm applied to elliptic curve cryptosystems*, IEEE Trans. Inform. Theory, 45 (1999), pp. 1717–1718.

[18] S. Galbraith, *Supersingular curves in cryptography*, in Advances in Cryptology—AsiaCrypt 2001, Lecture Notes in Comput. Sci. 2248, Springer-Verlag, Berlin, 2001, pp. 495–513.
[19] S. Galbraith, K. Harrison, and D. Soldera, *Implementing the Tate-pairing*, in Proceedings of the Fifth Algorithmic Number Theory Symposium, Lecture Notes in Comput. Sci. 2369, Springer-Verlag, Berlin, 2002, pp. 324–327.
[20] P. Gemmell, *An introduction to threshold cryptography*, in CryptoBytes, a technical newsletter of RSA Laboratories, 2 (1997), pp. 7–12.
[21] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, *Robust and efficient sharing of RSA functions*, J. Cryptology, 13 (2000), pp. 273–300.
[22] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, *Secure distributed key generation for discrete-log based cryptosystems*, in Advances in Cryptology—Eurocrypt 1999, Lecture Notes in Comput. Sci. 1592, Springer-Verlag, Berlin, 1999, pp. 295–310.
[23] O. Goldreich, B. Pfitzmann, and R. Rivest, *Self-delegation with controlled propagation -or- What if you lose your laptop*, in Advances in Cryptology—Crypto 1998, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, Berlin, 1998, pp. 153–168.
[24] S. Goldwasser and S. Micali, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.
[25] D. Hühnlein, M. Jacobson, and D. Weber, *Towards practical non-interactive public key cryptosystems using non-maximal imaginary quadratic orders*, in Selected Areas in Cryptography, Lecture Notes in Comput. Sci. 2012, Springer-Verlag, Berlin, 2000, pp. 275–287.
[26] A. Joux, *A one round protocol for tripartite Diffie-Hellman*, in Proceedings of the Fourth Algorithmic Number Theory Symposium, Lecture Notes in Comput. Sci. 1838, Springer-Verlag, Berlin, 2000, pp. 385–394.
[27] A. Joux, *The Weil and Tate pairings as building blocks for public key cryptosystems*, in Proceedings of the Fifth Algorithmic Number Theory Symposium, Lecture Notes in Comput. Sci. 2369, Springer-Verlag, Berlin, 2002, pp. 20–32.
[28] A. Joux and K. Nguyen, *Separating Decision Diffie-Hellman from Diffie-Hellman in Cryptographic Groups*, http://eprint.iacr.org, 2001.
[29] S. Lang, *Elliptic Functions*, Addison–Wesley, Reading, MA, 1973.
[30] U. Maurer, *Towards proving the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms*, in Advances in Cryptology—Crypto 1994, Lecture Notes in Comput. Sci. 839, Springer-Verlag, Berlin, 1994, pp. 271–281.
[31] U. Maurer and Y. Yacobi, *Non-interactive public-key cryptography*, in Advances in Cryptology—Crypto 1991, Lecture Notes in Comput. Sci. 547, Springer-Verlag, Berlin, 1991, pp. 498–507.
[32] A. Menezes, T. Okamoto, and S. Vanstone, *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Trans. Inform. Theory, 39 (1993), pp. 1639–1646.
[33] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL, 1996.
[34] V. Miller, *Short Programs for Functions on Curves*, manuscript.
[35] A. Miyaji, M. Nakabayashi, and S. Takano, *New explicit condition of elliptic curve trace for FR-reduction*, IEICE Trans. Fundamentals, E84 A (2001), pp. 1234–1243.
[36] P. Paillier and M. Yung, *Self-escrowed public-key infrastructures*, in Information Security and Cryptology—ICISC 1999, Lecture Notes in Comput. Sci. 1787, Springer-Verlag, Berlin, 1999, pp. 257–268.
[37] C. Rackoff and D. Simon, *Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack*, in Advances in Cryptology—Crypto 1991, Lecture Notes in Comput. Sci. 547, Springer-Verlag, Berlin, 1991, pp. 433–444.
[38] R. Rivest, A. Shamir, and D. Wagner, *Time Lock Puzzles and Timed Release Cryptography*, Technical report MIT/LCS/TR-684, http://www.lcs.mit.edu/publications/.
[39] K. Rubin and A. Silverberg, *Supersingular abelian varieties in cryptography*, in Advances in Cryptology—Crypto 2002, Lecture Notes in Comput. Sci. 2140, Springer-Verlag, Berlin, 2002, pp. 336–353.
[40] R. Sakai, K. Ohgishi, and M. Kasahara, *Cryptosystems Based on Pairings*, in Proceedings of the Symposium on Cryptography and Information Security, Technical group on Information Security of the Institute of Electronics, Information and Communication Engineers (IEICE), Oiso, Japan, 2001, pp. 26–28.
[41] A. Shamir, *Identity-based cryptosystems and signature schemes*, in Advances in Cryptology—Crypto 1984, Lecture Notes in Comput. Sci. 196, Springer-Verlag, Berlin, 1984, pp. 47–53.
[42] V. Shoup, *Lower bounds for discrete logarithms and related problems*, in Proceedings of Eurocrypt 1997, Lecture Notes in Comput. Sci. 1233, Springer-Verlag, Berlin, 1997, pp. 256–266.

[43] J. Silverman, *The Arithmetic of Elliptic Curve*, Springer-Verlag, Berlin, 1986.
[44] S. Tsuji and T. Itoh, *An ID-based cryptosystem based on the discrete logarithm problem*, IEEE J. Selected Areas in Communication, 7 (1989), pp. 467–473.
[45] H. Tanaka, *A realization scheme for the identity-based cryptosystem*, in Advances in Cryptology—Crypto 1887, Lecture Notes in Comput. Sci. 293, Springer-Verlag, Berlin, 1987, pp. 341–349.
[46] E. Verheul, *Evidence that XTR is more secure than supersingular elliptic curve cryptosystems*, in Advances in Cryptology—Eurocrypt 2001, Lecture Notes in Comput. Sci. 2045, Springer-Verlag, Berlin, 2001, pp. 195–210.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 616–642

3-DIMENSIONAL EUCLIDEAN VORONOI DIAGRAMS OF LINES WITH A FIXED NUMBER OF ORIENTATIONS*

VLADLEN KOLTUN† AND MICHA SHARIR‡

Abstract. We show that the combinatorial complexity of the Euclidean Voronoi diagram of n lines in $\mathbb{R}^3$ that have at most c distinct orientations is $O(c^3 n^{2+\varepsilon})$ for any $\varepsilon > 0$. This result is a step toward proving the long-standing conjecture that the Euclidean Voronoi diagram of lines in three dimensions has near-quadratic complexity. It provides the first natural instance in which this conjecture is shown to hold. In a broader context, our result adds a natural instance to the (rather small) pool of instances of general 3-dimensional Voronoi diagrams for which near-quadratic complexity bounds are known.

Key words. computational geometry, Voronoi diagrams, arrangements, lines in space

AMS subject classifications. 68U05, 52C45, 68Q25, 14P99, 51N20

PII. S0097539702408387

1. Introduction.

Background. The Voronoi diagram of a set Γ of disjoint objects ("sites") in some space under some metric is a subdivision of the space into cells, one cell per site, such that the cell associated with a site $O \in \Gamma$ comprises the points in space for which O is closer (under the given metric) than all other sites of Γ.

The study of Voronoi diagrams in the plane has been very extensive over the past 20 years, and the structure of such diagrams is by now thoroughly understood. The study has covered diagrams for many kinds of sites, and for many kinds of metrics or distance functions, and has also considered other variants of the problem, such as kth order diagrams, constrained Delaunay triangulations, and more. Surveys of the state of the art are given in Aurenhammer and Klein [4] and Fortune [10].

In contrast, Voronoi diagrams in three and higher dimensions have been much less studied, and many basic problems are still wide open. Most variants of planar Voronoi diagrams have linear complexity, which is usually a consequence of the planarity of the diagram. In three dimensions, a prevailing conjecture is that the complexity of Voronoi diagrams should be in general at most quadratic or near-quadratic in the number of sites. This is known to hold only for a very few special cases, including the cases of point sites under the Euclidean metric [16, 21], point sites under any "polyhedral" metric or distance function (i.e., distance functions induced by a convex polytope with $O(1)$ facets; see [5, 15, 24] for details), line sites under similar distance functions [6], and sphere sites under the Euclidean metric [3]. Only very recently,

*Received by the editors May 28, 2002; accepted for publication (in revised form) October 23, 2002; published electronically March 5, 2003. Work on this paper has been supported by a grant from the Israel Science Fund (for a Center of Excellence in Geometric Computing). Work by the second author was also supported by NSF grants CCR-97-32101 and CCR-00-98246, by a grant from the U.S.–Israel Binational Science Foundation, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University. This work is part of the first author's Ph.D. dissertation, prepared under the supervision of the second author at Tel Aviv University. A preliminary version of this paper has appeared in the *Proceedings of the* 18*th ACM Symposium on Computational Geometry*, ACM, New York, 2002.

http://www.siam.org/journals/sicomp/32-3/40838.html

†Computer Science Division, University of California, Berkeley, CA 94720-1776 (vladlen@cs.berkeley.edu).

‡School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (sharir@cs.tau.ac.il).

the authors [17] have shown this to hold also in the case of arbitrary polyhedral sites under polyhedral distance functions.

In all the other, "open" cases, cubic or near-cubic upper bounds for the complexity of 3-dimensional Voronoi diagrams are known. They are a consequence of the representation of such diagrams as lower envelopes of trivariate functions, each measuring the distance from a point in $\mathbb{R}^3$ to one of the sites; see [8] for this representation and [23] for the bounds just stated. In contrast, only quadratic or near-quadratic lower bounds for the complexity of 3-dimensional diagrams are known [2, 6].

The case of the Euclidean metric appears to be harder than the case of polyhedral metrics (or distance functions), because the trivariate functions that measure distances are curved (except for the special case of point sites, where they can be transformed into linear functions), and the constraints that define the diagram are harder to analyze. The simplest open case of 3-dimensional Euclidean diagrams is that in which the sites are lines. This specific problem is listed as Problem 3 in the list of open problems in computational geometry, recently published by Mitchell and O'Rourke [19]. A recent result that lends credence to the conjecture that the complexity of such diagrams is near-quadratic is due to Agarwal and Sharir [1], who showed that the complexity of the union of n infinite congruent cylinders in 3-space is near-quadratic. The boundary of this union can be interpreted as a cross-section of the Euclidean Voronoi diagram of the axes of the cylinders, being the locus of all those points whose distance to the nearest axis has a fixed value (equal to the common radius of the cylinders). The complicated proof in [1] and the fact that the result applies merely to a single cross-section of the diagram suggest that the problem involving the whole Euclidean Voronoi diagram of lines might be particularly hard to tackle.

Our contribution. In this paper, we obtain the first result toward the described goal. We study the special case in which the sites are lines that have a fixed number c of distinct orientations (and the metric is Euclidean). Even this special case is quite nontrivial to analyze. We show that the complexity of the diagram is $O(c^3 n^{2+\varepsilon})$ for any $\varepsilon > 0$, where the constant of proportionality depends on ε. This implies, in particular, that when the number of distinct orientations in a collection of lines is constant (that is, $c = O(1)$), the complexity of its Euclidean Voronoi diagram is $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$. This completely confirms the above-mentioned conjecture in this case.

The motivation underlying the study of Voronoi diagrams in computational geometry has always been algorithmic. They provide a natural data structure for handling a variety of applications, important both in theory and in practice, such as proximity (nearest neighbor) queries, high-clearance placements and motion planning problems, clustering and classification problems, and many more (see, among others, the survey by Aurenhammer and Klein [4] and the book by Okabe et al. [20] for a description of many of these applications).

There are several general techniques for computing Voronoi diagrams, such as randomized incremental construction or sweep-based methods, and many more ad hoc approaches. However, a precursory stage to the design of any algorithm for computing Voronoi diagrams is obtaining sharp bounds on their complexity. This will serve as a lower bound for the efficiency of any such algorithm and quite often can be used in the design of algorithms with roughly the same running time. Nevertheless, most of the algorithmic study of Voronoi diagrams has been confined to planar diagrams for the good reason that we are still lacking sharp general bounds for the complexity of generalized 3-dimensional diagrams.

The results presented in this paper are an attempt to remedy this situation. The special case we treat is important because it provides us with one more problem instance where near-quadratic bounds can be established. We hope that the method developed here will find applications in the analysis of other types of 3-dimensional Voronoi diagrams (see the remark at the end of section 3) and thereby lead us further toward the ultimate goal of establishing near-quadratic bounds for general 3-dimensional diagrams, following which near-quadratic algorithms for their construction will not be too difficult to design.

Moreover, the considered setting of lines with a fixed number of orientations is interesting in its own right. It is applicable, for example, to the problem of motion planning, or of finding largest free placements, of a ball amid a collection of "beams" or "pipes" in 3-space. It is a natural assumption that the beams have only a constant number of orientations. (Typical examples of this setting occur in architectural design.)

Organization. We first study, in section 2, the special case in which the lines have at most three distinct orientations. In this special case, we obtain the slightly improved bound $O(n\lambda_5(n))$, where $\lambda_5(n) = O(n \cdot \alpha(n)^{O(\alpha(n))})$ is the maximum length of Davenport–Schinzel sequences of order 5 on n symbols, and where $\alpha(n)$ is the extremely slowly growing inverse Ackermann function (see [23] for details). The case of four orientations is treated in section 3, and the simple extension to more than four orientations is described in section 4.

2. The case of two or three orientations. Let L be a set of n lines in 3-space which have up to three distinct orientations. Thus L can be written as $R \cup B \cup G$, where all the lines in R (called "red" lines) have the same orientation, and the same holds for the lines of B ("blue" lines) and those of G ("green" lines).

We adopt a limited general position assumption on L as follows. First, we assume that each of the collections R, B, and G is in general position in the sense that its intersection with any fixed generic plane is a collection of points in general position (that is, it does not contain collinear triples or cocircular quadruples of points or other degenerate configurations). We also assume that the three vectors that are parallel to the orientations of the collections R, B, and G do not lie in a common plane.

Before we proceed, we need to mention some basic properties of *bisectors* and *trisectors* of lines, which are, respectively, the loci of points equidistant from two and three lines. These geometric properties are reported here without proofs, which are given as an appendix below, in order to maintain the flow of exposition. The main conclusions from the analysis carried out in the appendix are as follows. A bisector of two lines is in general a hyperbolic paraboloid, which is a doubly ruled quadratic surface. (It degenerates to a plane when the two lines are parallel.) A trisector of three pairwise nonparallel lines is an algebraic curve of degree four and, if nonsingular, has exactly four components, all unbounded. If two of the three defining lines are parallel, the trisector becomes a planar conic section (of degree two, consisting of at most two unbounded components). If all three lines are parallel, the trisector is a line parallel to them. The fact that no component of any trisector is bounded will be significant in our analysis. In what follows, we will denote the bisector of two lines e, f by $H_{e,f}$, and the trisector of three lines e, f, g will be denoted by $\tau_{e,f,g}$.

Denote the Euclidean Voronoi diagram of L by $Vor(L)$. We begin by bounding the number of its vertices. Let v be such a vertex, incident to the cells of four lines $\ell_1, \ell_2, \ell_3, \ell_4$. At least two of them must be of the same color. Suppose first that three of them are of the same color, say, $\ell_1, \ell_2, \ell_3 \in R$. Project v and all the lines of R

onto a plane π orthogonal to these lines. Then each line of R projects to a point, and v projects onto a vertex v^* of the planar Voronoi diagram of the projected points within π. The number of such vertex projections v^* is thus at most $2n-4$. Moreover, the number of vertices v that can project onto the same point v^* is at most $2n$. This is because the radius r of the ball centered at v and touching ℓ_1, ℓ_2, ℓ_3 is equal to the radius of the disk within π centered at v^* and touching the point projections of these three lines. As we slide a ball of radius r while maintaining contact with ℓ_1, ℓ_2, ℓ_3, we reach at most $2n$ placements where it touches a fourth line. Each of these touching placements in which the ball is not crossed by any other line gives rise to a Voronoi vertex that projects onto v^*. This implies that the overall number of Voronoi vertices of the kind under consideration is at most $(2n-4) \cdot 2n = O(n^2)$.

Suppose then that exactly two of the four lines are of the same color, say, $\ell_1, \ell_2 \in R$, $\ell_3 \in B$, and $\ell_4 \in G$. If we project v and the lines of R onto the same plane π as above, we obtain that the projection of v lies on a Voronoi edge of the planar diagram of the point projections of the red lines. The number of such edges is $O(n)$.

Fix such an edge e, and consider the 2-dimensional slab Σ_e obtained by sweeping e in the direction of the red lines; by construction, $v \in \Sigma_e$. Moreover, Σ_e is the locus of all the centers of balls that touch ℓ_1 and ℓ_2 and no other red line. Let H_e denote the plane containing Σ_e, and let ℓ_0 be the line of intersection between H_e and the plane π_0 spanned by ℓ_1 and ℓ_2—this intersection is the midline of the 2-dimensional slab spanned by ℓ_1 and ℓ_2. Denote the two halfspaces bounded by π_0 as π_0^+ and π_0^-. See Figure 1.

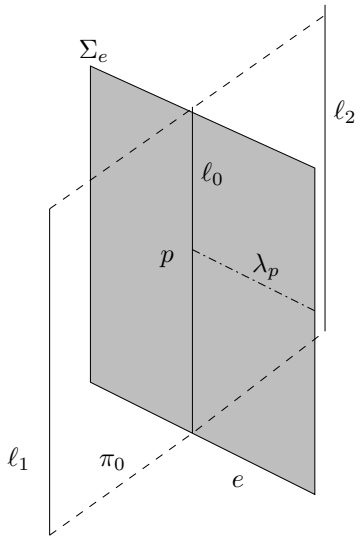


FIG. 1. *The bisector of ℓ_1 and ℓ_2.*

Fix a point $p \in \ell_0$, and consider the line λ_p that passes through p, lies in H_e, and is orthogonal to ℓ_0. Parametrize λ_p by a real parameter y, where $y = 0$ at p, $y > 0$ within π_0^+, and $y < 0$ within π_0^-. Move a point q along the entire λ_p in the direction of increasing y. The ball centered at q and touching ℓ_1, ℓ_2 has the property that its intersection with π_0^+ keeps expanding during the motion (i.e., any point of π_0^+ that the moving ball meets will remain inside the ball as its center keeps moving in the

above direction). Similarly, the portion of the moving ball within π_0^- keeps shrinking "into itself."

Let each line $\ell \in B \cup G$ define two rays $\ell^+ = \ell \cap \pi_0^+$, $\ell^- = \ell \cap \pi_0^-$. With each ray ℓ^+ (resp., ℓ^-), associate a function ψ_{ℓ^+} (resp., ψ_{ℓ^-}) on ℓ_0, where $\psi_{\ell^+}(p)$ (resp., $\psi_{\ell^-}(p)$) for $p \in \ell_0$ is the y-value of the center of the ball that touches ℓ_1, ℓ_2, and ℓ^+ (resp., ℓ^-), where the center lies on λ_p. The functions ψ_{ℓ^+}, ψ_{ℓ^-} are defined (and continuous) when ℓ does not intersect the disk centered at p, lying in π_0, and touching ℓ_1 and ℓ_2. Hence the (common) domain of definition of ψ_{ℓ^+} and ψ_{ℓ^-} is either the full line ℓ_0 if ℓ does not intersect the 2-dimensional slab spanned by ℓ_1 and ℓ_2 or the union of two rays along ℓ_0 otherwise.

Denote the collection of the functions ψ_{ℓ^+} (resp., ψ_{ℓ^-}) for $\ell \in B \cup G$ by Ψ^+ (resp., by Ψ^-). The preceding observations imply that any Voronoi vertex $v \in \Sigma_e$ under consideration (two of whose defining lines are in $B \cup G$) corresponds either to a vertex of the lower envelope of Ψ^+ or to a vertex of the upper envelope of Ψ^- or to an intersection point between the two envelopes.

It is easily seen that any pair of functions of the above kind intersect in at most four points. Indeed, any such intersection point w is equidistant from ℓ_1, ℓ_2, and from two other lines $\ell_3, \ell_4 \in B \cup G$. That is, we have

$$d^2(w, \ell_1) \left(= d^2(w, \ell_2)\right) = d^2(w, \ell_3) = d^2(w, \ell_4).$$

The squared distance of a point w from a line that passes through a point a and has unit direction u is

$$\|w - a\|^2 - ((w - a) \cdot u)^2,$$

which is a quadratic polynomial in the coordinates of w. Since w lies on the plane H_e, we obtain a system of two quadratic equations in two variables which has at most four solutions (see also the proof of Lemma 3.1 below).

It is shown, e.g., in [23, Lemma 1.8] that the complexity of the upper or lower envelope of continuous functions, so that each function is defined on a ray or on the whole real line, and so that each pair of them intersect in at most four points, is $O(\lambda_5(n)) = O(n \cdot \alpha(n)^{O(\alpha(n))})$ [23], where $\lambda_5(n)$ is the maximum length of Davenport–Schinzel sequences of order 5 on n symbols, and where $\alpha(n)$ is the extremely slowly growing inverse Ackermann function. As observed above, we can split each partially defined function in $\Psi^+ \cup \Psi^-$ into two functions, each defined over a ray. We thus conclude that the number of Voronoi vertices in (the relative interior of) Σ_e is $O(\lambda_5(n))$. Multiplying this bound by the number $O(n)$ of edges e and adding the preceding bound $O(n^2)$ on the number of vertices defined by three lines of the same color, we conclude that the number of vertices of the diagram $Vor(L)$ is $O(n\lambda_5(n))$.

We next bound the number of edges of $Vor(L)$. If an edge e is delimited by a Voronoi vertex v, we charge e to v. By the general position assumption, each v is charged at most four times, so the number of edges e of this kind is $O(n\lambda_5(n))$. Let e be a Voronoi edge that has no incident Voronoi vertex. As mentioned above, the analysis of trisectors implies that e is not bounded.

Fix two planes $\pi^{\pm}$: $z = \pm z_0$ such that each unbounded edge of $Vor(L)$ intersects at least one of them. (Assuming that the coordinate directions are generic, such planes exist.) It therefore suffices to bound the complexity of the cross-sections of $Vor(L)$ with the planes $\pi^{\pm}$. Consider, say, the plane π^+. The Voronoi cells in each of the monochromatic diagrams $Vor(R)$, $Vor(B)$, $Vor(G)$ are unbounded convex prisms, whose faces are all parallel to the orientation of the respective collection of lines, and

the overall complexity of each diagram is $O(n)$. Hence, the intersection of π^+ with each of these monochromatic diagrams is a planar convex subdivision of complexity $O(n)$. The overlay of these cross-sections is a planar convex subdivision of complexity $O(n^2)$. For each cell ξ of the overlay, there exist a fixed red line r, a fixed blue line b, and a fixed green line g, which are the nearest red, blue, and green lines to any point in ξ, respectively. It follows that the complexity of the overall diagram $Vor(L)$ within ξ is bounded by a constant, which implies that the complexity of the diagram within π^+ (and, symmetrically, within π^-) is $O(n^2)$.

This implies that the number of unbounded edges of $Vor(L)$ is $O(n^2)$. It is easily seen that the number of 2-faces of the diagram is proportional to the number of vertices plus the number of edges plus $O(n^2)$. Finally, the number of 3-cells is only n: Each line has a connected, star-shaped Voronoi cell [18]. Hence we obtain the following theorem, the main result of this section.

THEOREM 2.1. *The complexity of the Voronoi diagram of a set of n lines with at most three distinct orientations is $O(n\lambda_5(n)) = O(n^2 \cdot \alpha(n)^{O(\alpha(n))})$.*

3. The case of four orientations. We now assume that the given set L of lines is the union of four subsets, each consisting of lines at a fixed direction. We denote these subsets by R (consisting of "red" lines), B (consisting of "blue" lines), G (consisting of "green" lines), and Y (consisting of "yellow" lines). The proof of the following elementary geometric fact is provided for completeness.

LEMMA 3.1. *The maximum number of balls tangent to four given lines in 3-space, assuming general position, is 8.*

Proof. As already noted, the distance $d(\mathbf{x}, \ell)$ between a point $\mathbf{x} \in \mathbb{R}^3$ and a line ℓ, passing through a point a and having unit direction u, satisfies

$$d^2(\mathbf{x}, \ell) = \|\mathbf{x} - a\|^2 - ((\mathbf{x} - a) \cdot u)^2,$$

which is a quadratic function of $\mathbf{x}$. Given four lines $\ell_1, \ell_2, \ell_3, \ell_4$ in general position, the center $\mathbf{x}$ of a ball that is tangent to all four lines has to satisfy the equations

$$d^2(\mathbf{x}, \ell_1) = d^2(\mathbf{x}, \ell_2) = d^2(\mathbf{x}, \ell_3) = d^2(\mathbf{x}, \ell_4).$$

These are three quadratic equations, so, by Bezout's theorem [14], the number of solutions is at most $2^3 = 8$.

The number 8 can be attained: We first give a construction where the lines are not in general position. Take ℓ_1, ℓ_2, ℓ_3 to be any three nonconcurrent lines in the xy-plane. They determine four disks D_1, D_2, D_3, D_4 in that plane that are tangent to all three of them, as shown in Figure 2. Take ℓ_4 to be any line perpendicular to the xy-plane, meeting the plane at a point not lying in any of these disks. Fix a disk D_i, and let λ_i be the z-vertical line passing through the center of D_i; this is the locus of all centers of balls that touch ℓ_1, ℓ_2, ℓ_3 and meet the xy-plane at D_i. It is easily seen that there are exactly two points on λ_i, symmetric to each other with respect to the xy-plane, that are centers of balls that also touch ℓ_4. For any specific disc D_i, this yields two distinct balls that touch all four lines, giving us eight such balls overall. By slightly perturbing the lines, we can obtain a construction for lines in general position. This completes the proof of the lemma. □

Let $\ell_1, \ell_2, \ell_3, \ell_4$ be four given lines of different colors. Let $s \leq 8$ denote the number of balls tangent to all four of them, and let $c_1, \ldots, c_s$ denote the centers of these balls, sorted in increasing order of their x-coordinate. (The coordinate frame is assumed to be generic so that no two c_i's have the same x-coordinates.) Define the *index* $\mathrm{ind}(c_i)$ of c_i to be $\min\{i - 1, s - i\}$, so we have $0 \leq \mathrm{ind}(c_i) \leq 3$ for each i.

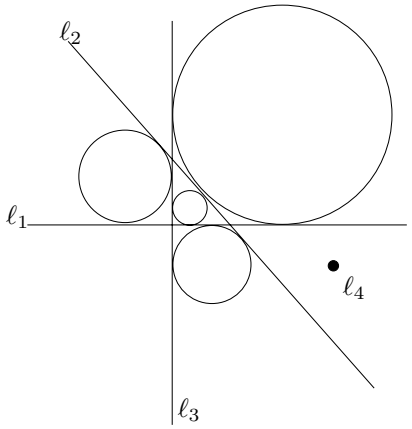


FIG. 2. *Four lines having eight Voronoi vertices.*

With each line $\ell \in L = R \cup B \cup G \cup Y$ we associate the squared distance function $f_\ell \colon \mathbb{R}^3 \mapsto \mathbb{R}$, given by $f_\ell(\mathbf{x}) = d^2(\mathbf{x}, \ell)$. Let $\mathcal{E}_\mathcal{F}$ denote the lower envelope of the set $\mathcal{F} = \mathcal{F}(L) = \{f_\ell \mid \ell \in L\}$. Clearly, the *minimization diagram* of $\mathcal{E}_\mathcal{F}$, namely, the projection of (the graph of) $\mathcal{E}_\mathcal{F}$ onto the xyz-space, is the Voronoi diagram $Vor(L)$ (see also [8]).

For each point $\mathbf{q} = (q_1, q_2, q_3, q_4) \in \mathbb{R}^4$, define its *$R$-level* (resp., *$B$-level*, *$G$-level*, *$Y$-level*) to be the number of lines $\ell \in R$ (resp., $\ell \in B$, $\ell \in G$, $\ell \in Y$) whose corresponding function graphs pass below $\mathbf{q}$; that is, $q_4 > f_\ell(q_1, q_2, q_3)$. The *combined level* of $\mathbf{q}$ is the sum of its red, blue, green, and yellow levels. We denote the graph of each $f_\ell \in \mathcal{F}$ by $\tilde{f}_\ell$. Denote by $\tilde{R}$ the collection of all graphs $\tilde{f}_\ell$ for $\ell \in R$, and define $\tilde{B}$, $\tilde{G}$, $\tilde{Y}$, and $\tilde{L}$ analogously. Let $\mathcal{A}(\tilde{L})$ denote the arrangement in $\mathbb{R}^4$ of the graphs $\tilde{f}_\ell$ of the functions in $\tilde{L}$. Clearly, for a vertex $\mathbf{q}$ of $\mathcal{A}(\tilde{L})$, $\mathbf{q}$ is a vertex of $\mathcal{E}_\mathcal{F}$ if and only if the combined level of $\mathbf{q}$ is 0.

Let $V_0^{(j)}(L)$ (resp., $V_{\leq k}^{(j)}(L)$) denote the number of "4-colored" vertices $\mathbf{q}$ of $\mathcal{A}(\tilde{L})$ (i.e., vertices incident to a red graph, a blue graph, a green graph, and a yellow graph) of index $\leq j$, whose combined level is 0 (resp., at most k). Put $V_0(L) = V_0^{(3)}(L)$ and $V_{\leq k}(L) = V_{\leq k}^{(3)}(L)$. We also put $V_0^{(j)}(n) = \max_L V_0^{(j)}(L)$, where the maximum is taken over all families L of n lines, each having one of the four given orientations; $V_{\leq k}^{(j)}(n)$ is defined analogously. Using the Clarkson–Shor bound on levels [7], we have

$$V_{\leq k}^{(j)}(n) = O\left(k^4 V_0^{(j)}\left(\frac{n}{k}\right)\right).$$

As mentioned in section 2 and proven in the appendix, every connected component of any trisector is unbounded. However, in the proof below, we will *not* make use of this property at all. This will be significant when we extend the analysis to more general setups—see a discussion at the end of this section.

3.1. Irregular vertices. Let v be a 4-colored vertex of the diagram, interpreted as a vertex of the lower envelope $\mathcal{E}_\mathcal{F}$, incident to four graphs $\tilde{f}_r$, $\tilde{f}_b$, $\tilde{f}_g$, $\tilde{f}_y$ for some $r \in R$, $b \in B$, $g \in G$, and $y \in Y$. The vertex v is incident to four edges of the envelope, which we denote mnemonically as rbg, rby, rgy, and bgy, where $rbg \subseteq \tau_{r,b,g}$ denotes the edge lying on the graphs $\tilde{f}_r$, $\tilde{f}_b$, $\tilde{f}_g$, and similarly for the three other edges. As noted in [22], at least one of these edges emanates from v in the positive x-direction,

and at least one edge emanates in the negative x-direction. We call v a *regular* vertex if exactly two of these edges emanate from v in the positive x-direction and exactly two emanate from v in the negative x-direction. Otherwise, we call v *irregular*.

LEMMA 3.2. *There are only $O(n\lambda_5(n))$ irregular vertices.*

Proof. Let v be an irregular vertex. If v is not 4-colored, then the claim follows from Theorem 2.1, so assume that v is 4-colored, and use the above notation to denote the surfaces and edges incident to v. Suppose, without loss of generality, that three of the incident edges emanate from v to the left, and assume that they are rbg, rby, and rgy. In this case (assuming general position), v is a locally x-maximal vertex of the Voronoi cell $V(r)$ of r. Clearly, each line has a single connected Voronoi cell. In fact, each cell, star-shaped with respect to its defining line, is also simply connected; see, e.g., [18].

As shown, e.g., in [12, Lemma 2.4], the number of locally x-extremal points of a simply connected 3-dimensional region K is proportional to 1 plus the number of *critical points* of ∂K (relative to the x-direction). These are points w for which the cross-section of the interior of K with the yz-parallel plane through w is disconnected near w but becomes connected (near w) when the plane slightly translates in some direction. Hence the number of irregular vertices of $Vor(L)$ is proportional to the number of critical points of cell boundaries plus $O(n)$.

Assuming general position, each critical point w of $\partial V(r)$ is incident to only three surfaces; it is typically a locally x-extremal point of a Voronoi edge of $V(r)$. Suppose, without loss of generality, that w is incident to $\tilde{f}_r, \tilde{f}_{b_1}, \tilde{f}_{g_1}$ for some $b_1 \in B$, $g_1 \in G$. Then w is a locally x-extremal point of (the relative interior of) a Voronoi edge (a portion of τ_{r,b_1,g_1}) of the 3-colored Voronoi diagram $Vor(R \cup B \cup G)$. By Theorem 2.1, the overall number of such features is $O(n\lambda_5(n))$, and this completes the proof of the lemma. □

3.2. The counting scheme. In light of Lemma 3.2, this section is devoted to bounding the number of regular vertices of $Vor(L)$. This number is estimated using a variation of the "counting scheme" technique, as introduced by Halperin and Sharir [11, 22] (see also [23]).

Let v be a 4-colored regular vertex, incident to $\tilde{f}_r, \tilde{f}_b, \tilde{f}_g, \tilde{f}_y$, using the notation introduced above. Let $0 \le j \le 3$ be the index of v. Without loss of generality, assume that there are exactly j vertices incident to $\tilde{f}_r, \tilde{f}_b, \tilde{f}_g, \tilde{f}_y$ to the *right* (that is, in the x-increasing direction) of v. By definition, v is incident to two edges of $\mathcal{E}_{\mathcal{F}}$ that emanate from it to the right, and to two edges that emanate from it to the left. Without loss of generality, assume that the edges emanating to the right are rbg and rby and the edges emanating to the left are rgy and bgy.

Consider the 2-dimensional bisector $H_{g,y}$. Denote by R_{gy} the set of trisectors $\tau_{g,y,r'}$ drawn as curves along $H_{g,y}$ for red lines $r' \in R$. Define in an analogous manner the sets B_{gy}, G_{gy}, and Y_{gy} (where the latter two sets exclude the ill-defined trisectors induced by g and y themselves). Let $\mathcal{A}_{gy}$ denote the 2-dimensional arrangement of the collection $R_{gy} \cup B_{gy} \cup G_{gy} \cup Y_{gy}$ of curves within $H_{g,y}$. It follows that there exists a face of $\mathcal{A}_{gy}$ that is also a 2-face of $\mathcal{E}_{\mathcal{F}}$ on $H_{g,y}$, such that v is a locally x-maximal vertex of that face.

Let $\gamma_r \in R_{gy}$ (resp., $\gamma_b \in B_{gy}$) denote the trisector $\tau_{r,g,y}$ (resp., $\tau_{b,g,y}$), regarded as a curve within $H_{g,y}$. If we follow γ_r from v to the right, we lie, locally near v, above $\mathcal{E}_{\mathcal{F}}$ (actually, above $\tilde{f}_b$), and similarly for γ_b (which lies locally above $\tilde{f}_r$). See Figure 3.

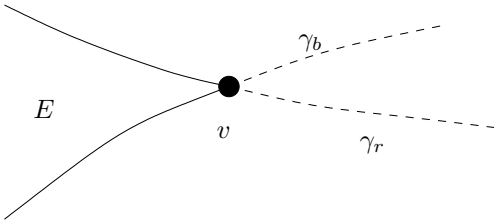


FIG. 3. *The vertex v on $H_{g,y}$—a view from the bottom (in $\mathbb{R}^4$).*

3.2.1. Initial counting stages and vertices of index 0 and 1.

LEMMA 3.3. *$V_0^{(0)}(n)$ and $V_0^{(1)}(n)$ are bounded by $O(n\lambda_5(n))$.*

Proof. Trace the curve γ_r from v to the right, and stop as soon as we reach one of the following *critical events* along γ_r:

(a) We reach another intersection of the four graphs $\tilde{f}_r, \tilde{f}_b, \tilde{f}_g, \tilde{f}_y$.
(b) We reach a 3-colored vertex.
(c) We reach $x = +\infty$.
(d) We reach a locally x-extremal point of the curve γ_r.

We refer to events of types (b)–(d) as *terminal events*.

Perform a similar tracing along γ_b. Suppose that at least one of the tracings, say, along γ_r, reaches a terminal event. The first such event either is a vertex of the 3-colored Voronoi diagram of $R \cup G \cup Y$ or can be charged to an edge of this diagram. By Theorem 2.1, the number of such events is thus $O(n\lambda_5(n))$, and each such event is uniquely counted by some vertex v. (This follows since between v and the terminal event we are always above $\mathcal{E}_{\mathcal{F}}$.) Hence the number of vertices v that fall in this case is $O(n\lambda_5(n))$. In particular, this bounds the number of vertices of index 0.

We may thus assume that the tracing of γ_r ends at a vertex u, and the tracing of γ_b ends at a vertex w, so that both u and w are incident to $\tilde{f}_r, \tilde{f}_b, \tilde{f}_g, \tilde{f}_y$ (see Figure 4). Moreover, the portion $\delta_r^{(1)}$ of γ_r between v and u and the portion $\delta_b^{(1)}$ of γ_b between v and w are both x-monotone, and neither of them contains a 3-colored vertex or another terminal event. In particular, u and w lie to the right of v, the red, green, and yellow levels of u are all 0, and the blue, green, and yellow levels of w are all 0.

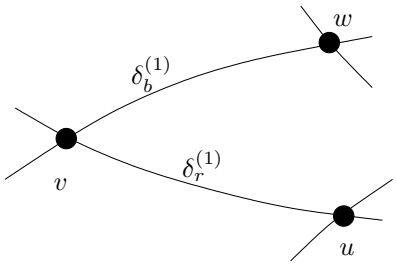


FIG. 4. *Tracing from v to the right.*

If $u = w$, then this is a vertex of the diagram (because all its colored levels are 0) of index at most $j - 1$ (because it lies to the right of v). The number of vertices v in this subcase is thus at most $V_0^{(j-1)}(n)$. In particular, this is easily seen to imply that

the number of vertices of index 1 is $O(n\lambda_5(n))$. We have thus shown the following:

$$V_0^{(0)}(n) = O(n\lambda_5(n)), \qquad V_0^{(1)}(n) = O(n\lambda_5(n)),$$

which completes the proof of the lemma. □

3.2.2. Subsequent counting stages and vertices of index 2. In what follows, we assume that $j = 2$ or 3. In light of the arguments made in the proof of Lemma 3.3, we may assume that $u \neq w$. Fix some threshold parameter k to be determined later.

LEMMA 3.4. *$V_0^{(2)}(n)$ is bounded by $O(k^3 n\lambda_5(n) + k^2 V_0(\frac{n}{5k}))$.*

Proof. Suppose that the blue (and thus the combined) level of u is at most $4k$. In this case, we charge v to u. The charging is unique, implying that the number of vertices v in this case is at most

$$V_{\leq 4k}^{(j-1)}(n) = O\left(k^4 V_0^{(j-1)}\left(\frac{n}{4k}\right)\right),$$

where, as already mentioned, we use the Clarkson–Shor bound on levels [7]. A similar charging is applied if the level of w is at most $4k$. Hence, in what follows, we may assume that $u \neq w$ and that both lie at combined level $> 4k$.

Let W denote the portion of $H_{g,y}$ consisting of all points that lie above the graphs of both f_r and f_b, and let W_0 be the connected component of W whose boundary contains v. The region W_0 is bounded, locally near v and to its right, by the two arcs $\delta_r^{(1)}$ and $\delta_b^{(1)}$, and v is a locally leftmost (x-minimal) vertex of W_0. Let $\delta_b^{(2)}$ (resp., $\delta_r^{(2)}$) denote the other edge of ∂W_0 incident to u (resp., to w). Both $\delta_r^{(1)}$ and $\delta_r^{(2)}$ are contained in the trisector $\tau_{r,g,y}$, although they do not have to lie on the same component of that curve. Similarly, $\delta_b^{(1)}$ and $\delta_b^{(2)}$ are contained in $\tau_{b,g,y}$. Without loss of generality, we assume that $\delta_r^{(1)}$ lies clockwise to $\delta_b^{(1)}$ (when viewed from above); see Figure 5 for an illustration of several possible shapes of W_0.

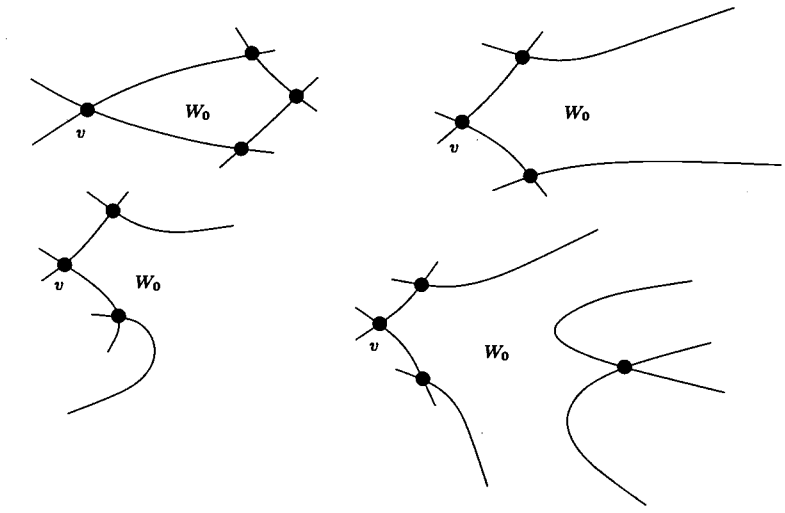


FIG. 5. *Several possible structures of the region W_0. In all cases, at most three vertices of W_0 lie to the right of v.*

Let ζ be a vertex of $\mathcal{A}_{gy}$ along $\delta_r^{(1)}$, incident to the graph of some other blue function f_β. (Recall that, by assumption, all vertices along $\delta_r^{(1)}$ are 4-colored.) Consider the trisector $\tau_{\beta,g,y}$ as a curve γ_β within $H_{g,y}$, and let δ_β denote the connected component of $\gamma_\beta \cap W_0$ incident to ζ. We say that δ_β is a *deep* arc if it contains at least k vertices of $\mathcal{A}_{gy}$. If δ_β is not deep and it contains a terminal event (namely, it contains a 3-colored vertex, or contains a locally x-extremal point, or reaches $x = \pm\infty$), we call it a *terminal* arc. Otherwise, we call it *shallow*. See Figure 6 for a special case of a shallow arc. Similar notation applies to red arcs that emanate from vertices of $\mathcal{A}_{gy}$ along $\delta_b^{(1)}$.

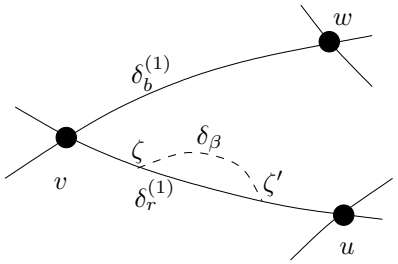


FIG. 6. *A shallow arc that lands back on* $\delta_r^{(1)}$.

Consider the first $4k$ vertices along $\delta_r^{(1)}$. (By assumption, $\delta_r^{(1)}$ must contain at least this many vertices.) If at least $2k$ of the corresponding arcs δ_β are deep, then collecting the first k vertices along each of these arcs yields a set of at least $2k^2$ vertices of $\mathcal{A}_{gy}$ within W_0, all lying at combined level at most $5k$. We claim that each of them is charged by vertices like v at most a constant number of times. Indeed, let η be such a vertex, lying on a deep blue arc δ_β. Note that the starting point ζ of δ_β is at red level 0, but all points in the relative interior of δ_β have strictly positive red levels. Hence we can trace δ_β back from η (there are two possible directions for this tracing) until we reach the first point ζ at red level 0. The point ζ must lie on $\delta_r^{(1)}$, and we can trace $\delta_r^{(1)}$ from ζ backward (to the left) until we reach v—the first vertex at combined level 0. Hence, using [7], as above, the number of vertices v in this subcase is at most

$$O\left(\frac{1}{k^2} V_{\le 5k}(n)\right) = O\left(k^2 V_0\left(\frac{n}{5k}\right)\right).$$

The same bound applies to the number of vertices v for which at least $2k$ of the first $4k$ vertices along $\delta_b^{(1)}$ are sources of deep red arcs.

Hence we may assume that, among the first $4k$ vertices along $\delta_r^{(1)}$, at least $2k$ are sources of shallow or terminal arcs, and similarly for $\delta_b^{(1)}$. If any of these arcs is terminal, we charge v to the corresponding terminal event along the arc. We note that such an event η lies at combined level at most $5k$. Hence η is or can be charged to a $(\le 5k)$-level feature of one of the 4-dimensional 3-colored arrangements $\mathcal{A}(\tilde{B} \cup \tilde{G} \cup \tilde{Y})$, $\mathcal{A}(\tilde{R} \cup \tilde{G} \cup \tilde{Y})$. Moreover, arguing as in the preceding paragraph, η is charged by vertices like v at most twice. By Theorem 2.1 and [7], the number of such events η, and thus also the number of vertices v that fall into this subcase, is at most

$$O\left(k^4 \cdot \frac{n}{5k} \lambda_5\left(\frac{n}{5k}\right)\right) = O(k^2 n \lambda_5(n)).$$

Hence we may assume that at least $2k$ of the first $4k$ vertices along $\delta_r^{(1)}$ are sources of shallow arcs, and none of these vertices are sources of terminal arcs. Moreover, the same property holds for $\delta_b^{(1)}$.

Suppose that one of these shallow arcs, δ_β, emanating from $\delta_r^{(1)}$, terminates also on $\delta_r^{(1)}$, as in Figure 6. By definition, δ_β does not encounter any blue graph $\tilde{f}_{\beta'}$ (for then δ_β would contain a 3-colored vertex and thus would be terminal). Hence the blue level of the terminal endpoint ζ' of δ_β is equal to the blue level of the starting point ζ, and all other levels of both endpoints are 0. In this case, we skip the portion of $\delta_r^{(1)}$ between ζ and ζ'. More precisely, we modify the tracing procedure used so far as follows: Trace $\delta_r^{(1)}$ to the right, starting from v, and attempt to collect either $2k$ deep arcs or a terminal arc or $2k$ shallow arcs that do not terminate on $\delta_r^{(1)}$. If during this tracing we reach a shallow arc δ_β that does terminate on $\delta_r^{(1)}$, we take a "shortcut" along δ_β and continue the tracing of $\delta_r^{(1)}$ from the other endpoint of δ_β. It is clear that this modified process must terminate successfully, or else we would reach the endpoint u of $\delta_r^{(1)}$, which then would lie at level $\leq 4k$, contrary to assumption. From now on, we apply a similarly modified tracing procedure to $\delta_b^{(1)}$ as well.

We thus reach the following situation. We have collected at least $2k$ shallow blue arcs that emanate from $\delta_r^{(1)}$ and terminate on other red edges of ∂W_0 and at least $2k$ shallow red arcs that emanate from $\delta_b^{(1)}$ and terminate on other blue edges of ∂W_0. The combined level of any point on any of these arcs is at most $5k$.

Suppose that one of the shallow blue arcs δ_β that emanates from $\delta_r^{(1)}$ terminates on a (red) edge $\delta_r^{(3)}$ of ∂W_0 that does not intersect $\tau_{b,g,y}$ at all. That is, $\delta_r^{(3)}$ is a full (bounded or unbounded) component of the trisector $\tau_{r,g,y}$, which lies fully above the graph of f_b, as in Figure 7. Let η be the "landing point" of δ_β on $\delta_r^{(3)}$. The combined level of η is at most $5k$. Trace $\delta_r^{(3)}$ from η to the right (i.e., in the positive x-direction) until we reach a terminal event η', to which we charge v. (Such an η' always exists: even if we do not encounter any finite event, we will reach $x = +\infty$, which is a terminal event, by definition.) Note that η' is or can be charged to a feature of the 4-dimensional arrangement $\mathcal{A}(\tilde{R} \cup \tilde{G} \cup \tilde{Y})$, whose combined level (in this 3-colored arrangement) is at most $5k$. Arguing as above, the number of such events is $O(k^2 n \lambda_5(n))$. Here we cannot claim that η' is uniquely charged by v, but we can still bound the number of times η' is charged, as follows.

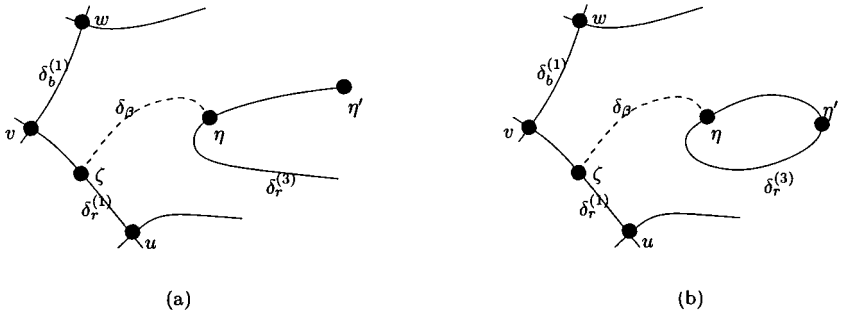


FIG. 7. *Charging v when a shallow arc lands on an edge of W_0 that does not meet other such edges.*

Trace $\delta_r^{(3)}$ back (in the negative x-direction) from η' (there may be two choices for r and for $\delta_r^{(3)}$ given a specific η', since η' may be a 3-colored vertex) until the first time we reach a point whose combined level (including the blue level) is at most $5k$. This backward tracing has to succeed: it will reach η or stop earlier. Then any charging vertex v must be a vertex incident to $\tilde{f}_r$, $\tilde{f}_g$, $\tilde{f}_y$, and to some $\tilde{f}_{b_1}$, from the at most $5k$ blue surfaces b_1 that lie below the stopping point. In other words, η' can be charged at most $O(k)$ times, implying that the number of vertices v that fall into this subcase is

$$O(k^3 n \lambda_5(n)). \tag{1}$$

A symmetric analysis applies if a shallow red arc lands on a blue edge of ∂W_0 which is a full component of $\tau_{b,g,y}$. Moreover, the analysis just given also holds if $\delta_r^{(3)}$ meets $\tilde{f}_b$ in only one of the two directions from η and extends to infinity in the other direction. It also holds if, in at least one of the two directions, we meet a terminal event before meeting $\tilde{f}_b$. And it also holds in the symmetric extended cases, in which the roles of the red and blue colors are interchanged.

The above analysis implies, in particular, that in what follows we may assume that none of the first $2k$ shallow arcs that emanate from $\delta_r^{(1)}$ and $\delta_b^{(1)}$ terminate on a bounded component of ∂W_0 that does not meet other components of ∂W_0. Note also that the analysis holds if $\delta_r^{(3)}$ is a bounded component of $\tau_{r,g,y}$ that lies fully to the right of v. Indeed, even if such a component does meet $\tilde{f}_b$, it must meet it at two points, both different from u, w and lying to the right of v, which is impossible.

Suppose now that one of the collected blue shallow arcs δ_β terminates on $\delta_r^{(2)}$, as in Figure 8. Each of the $\geq 2k$ red shallow arcs that we have collected along $\delta_b^{(1)}$ must cross δ_β. Indeed, none of these arcs terminate on $\delta_b^{(1)}$, by construction; they cannot terminate on $\delta_r^{(1)}$ or on $\delta_r^{(2)}$, for that would have made them terminal; and, as argued above, they also do not terminate on an isolated bounded component of ∂W_0. This, however, contradicts the shallowness of δ_β, since it cannot contain more than k crossings with other arcs. We have thus showed that none of the collected blue shallow arcs terminate on $\delta_r^{(2)}$. Symmetrically, it can be shown that none of the collected red shallow arcs terminate on $\delta_b^{(2)}$.

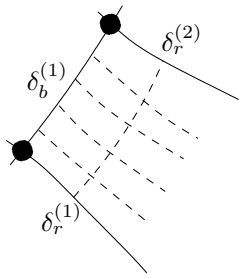


FIG. 8. *A shallow blue arc cannot "intercept," by terminating on $\delta_r^{(2)}$, the shallow red arcs emanating from $\delta_b^{(1)}$.*

The bounds accumulated so far account for all the vertices v with index at most 2. Specifically, we have

$$V_0^{(2)}(n) = O\left(k^3 n \lambda_5(n) + k^2 V_0\left(\frac{n}{5k}\right)\right),$$

thereby proving the lemma. □

3.2.3. Final counting stages and vertices of index 3.

LEMMA 3.5. *$V_0^{(3)}(n)$ is bounded by*

$$O\left(\left(k^2\ell^2+\ell^3\right)n\lambda_5(n)+V_0^{(2)}(n)+k^4V_0^{(2)}\left(\frac{n}{4k}\right)\right.$$

$$\left.+k^2V_0^{(3)}\left(\frac{n}{5k}\right)+\ell^4V_0^{(2)}\left(\frac{n}{5\ell}\right)+k^2\ell^2V_0^{(3)}\left(\frac{n}{6\ell}\right)\right).$$

Proof. From now on, we deal with vertices v of index 3. They are treated by considering a number of possible structures of the region W_0 as well as possible behavior patterns of arcs inside W_0 and bounding the maximal number of vertices v in each case. This will often be performed by charging v to certain features in W_0.

We already have sufficient machinery to dispose of vertices v for which $\delta_r^{(2)}$ and $\delta_b^{(2)}$ meet at a common endpoint, as in Figure 9. The preceding arguments allow us to assume that there are no shallow arcs that connect $\delta_r^{(1)}$ to $\delta_r^{(2)}$, or $\delta_b^{(1)}$ to $\delta_b^{(2)}$, and that there are no shallow arcs that land on any bounded component of W_0 within the quadrangle formed by $\delta_r^{(1)}$, $\delta_b^{(1)}$, $\delta_r^{(2)}$, and $\delta_b^{(2)}$. This means that, in this case, unless u and w have level $O(k)$, we can either collect a terminal arc or at least $2k$ deep arcs when sliding from v as above. In other words, we can charge v (almost uniquely) either to $\Theta(k^2)$ low-level vertices (at level $O(k)$) within W_0 or to a low-level terminal event within W_0 or to some other vertex of W_0 (that is, to u or to w) which lies at level at most $4k$ and has a smaller index. Arguing as above, the number of vertices v that fall into this case is

$$O\left(k^3n\lambda_5(n)+k^2V_0\left(\frac{n}{5k}\right)+k^4V_0^{(2)}\left(\frac{n}{4k}\right)\right).$$

We may thus assume that $\delta_r^{(2)}$ and $\delta_b^{(2)}$ do not meet.

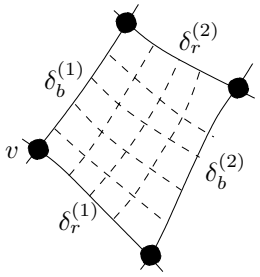


FIG. 9. *The case in which W_0 is a quadrangle.*

Suppose, without loss of generality, that the vertex u lies to the left of w. Then any shallow blue arc δ_β that emanates from $\delta_r^{(1)}$ must terminate at a point that lies to the right of v (regardless of whether it extends to the right or to the left); see Figure 10. This is due to the fact that these arcs are x-monotone. Let η be the terminal point of δ_β, and let $\delta_r^{(3)} \neq \delta_r^{(1)}, \delta_r^{(2)}$ denote the red edge of ∂W_0 that contains η. (The preceding analysis implies that we may assume that all shallow blue arcs that we have collected do land on a new red edge of ∂W_0.) Trace $\delta_r^{(3)}$ from η in the

increasing x-direction. (Note the two different situations that can arise, where we can turn from δ_β to the traced portion of $\delta_r^{(3)}$ either to the left or to the right.[1]) By the analysis just given, we may assume that this portion of $\delta_r^{(3)}$ terminates at a vertex t of W_0, incident to $\tilde{f}_r$, $\tilde{f}_b$, $\tilde{f}_g$, $\tilde{f}_y$, which lies to the right of v and is different from u, w. That is, t is the "missing" third sibling vertex of v that lies to the right of v. Moreover, the portion of $\delta_r^{(3)}$ between η and t contains no terminal event.

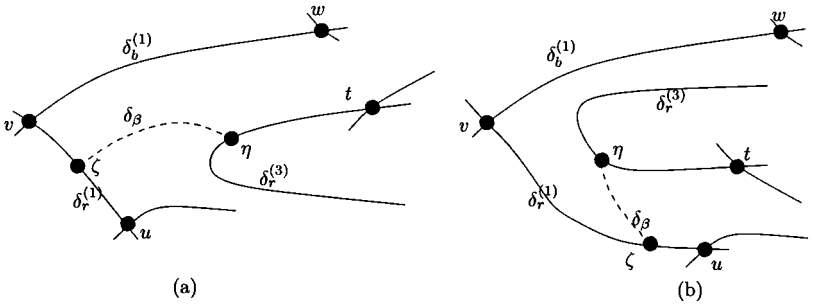


FIG. 10. *If u lies to the left of w, a shallow arc emanating from $\delta_r^{(1)}$ must terminate to the right of v. In* (a)*, we make a left turn from δ_β to $\delta_r^{(3)}$ at η, and in* (b)*, we make a right turn. In both cases, $\delta_r^{(3)}$ has to contain a vertex t of W_0 in the direction of our tracing.*

Suppose first that w lies to the left of t; see Figure 11. There must exist a red shallow arc δ_ρ that emanates from $\delta_b^{(1)}$ and does not cross δ_β (and it also cannot cross $\delta_r^{(1)}$, $\delta_r^{(2)}$, or $\delta_r^{(3)}$). Since δ_ρ is x-monotone, it must terminate at a point η' to the right of v, regardless of whether it extends to the right or to the left: the concatenation of $\delta_r^{(1)}$, δ_β, and $\delta_r^{(3)}$ up to t does not allow δ_ρ to reach points left of v because t lies to the right of w; see Figure 11. The point η' lies on some blue edge $\delta_b^{(3)} \neq \delta_b^{(1)}, \delta_b^{(2)}$. Tracing $\delta_b^{(3)}$ from η' in the positive x-direction, we may assume that it terminates at a vertex of W_0 (the case of a terminal event can be charged as above), which is necessarily t itself. Moreover, the portion of $\delta_b^{(3)}$ between η' and t contains no terminal event. We now note that the red and blue levels of t are both at most k since the red level of η and the blue level of η' are at most k and since there are no terminal events on $\delta_r^{(3)}$ between η and t and on $\delta_b^{(3)}$ between η' and t. Thus the combined level of t is $O(k)$. Since t is of index at most 1 (it lies to the right of w) and is uniquely charged by v, the number of vertices v in this subcase is $O(k^2 n \lambda_5(n))$.

Suppose then that w lies to the right of t. If any shallow red arc that emanates from $\delta_b^{(1)}$ and does not cross δ_β terminates to the right of v, we proceed as in the case, just treated, where t lies to the right of w. The only way in which this does not occur is when all these shallow red arcs emanate from $\delta_b^{(1)}$ in the negative x-direction, starting to the right of t and "bypassing" the concatenation of $\delta_r^{(1)}$, δ_β, and $\delta_r^{(3)}$ up to t. See Figure 12.

To handle this case, choose another threshold parameter $\ell \gg k$, to be determined later. If t lies at level at most $5k + 4\ell$, we charge v to t. We note, as above, that the

[1]Recall that, in the analysis of W_0, we refer to the view of this region from above (in the vertical direction of $\mathbb{R}^4$).

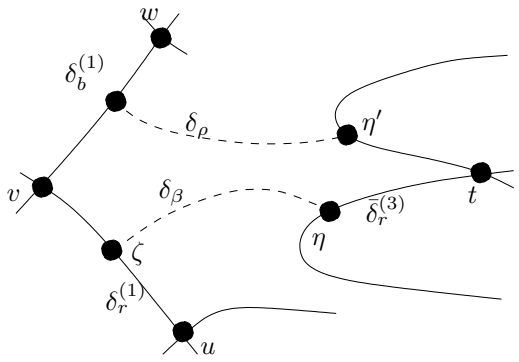


FIG. 11. *The case in which w lies to the left of t.*

charging is unique and use the fact that the index of t is at most 2 to conclude that the number of vertices v in this subcase is at most $V^{(2)}_{\leq 5k+4\ell}(n)$. Our choice of ℓ will ensure that $5k+4\ell \leq 5\ell$, so, using [7], the number of vertices v under consideration is

$$O\left(\ell^4 V_0^{(2)}\left(\frac{n}{5\ell}\right)\right).$$

Assume then that the level of t is $> 5k+4\ell$. Then the portion $\bar{\delta}_r^{(3)}$ of $\delta_r^{(3)}$ between η and t must contain at least 4ℓ (4-colored) vertices. We now apply a collection process for blue arcs that emanate from $\bar{\delta}_r^{(3)}$. The process is very similar to that applied to $\delta_r^{(1)}$ (and to $\delta_b^{(1)}$), except that we redefine the notions of being deep, terminal, or shallow in terms of the parameter ℓ rather than k. To distinguish between the old and new notions, we say that an arc is *ℓ-deep* (resp., *ℓ-shallow*) if it contains at least ℓ (resp., fewer than ℓ) vertices (and so that none of the first ℓ vertices is terminal). If one of the first ℓ vertices lying on an arc is terminal, the arc is said to be ℓ-terminal. The old notions are from now on designated, in complete analogy, as k-deep, k-shallow, and k-terminal.

The collection process on $\bar{\delta}_r^{(3)}$ is therefore as follows. Starting from η, we proceed along $\bar{\delta}_r^{(3)}$, taking shortcuts along ℓ-shallow arcs that land back on $\bar{\delta}_r^{(3)}$, and collect either 2ℓ ℓ-deep blue arcs or an ℓ-terminal blue arc or 2ℓ ℓ-shallow blue arcs that do not terminate on $\bar{\delta}_r^{(3)}$. The starting point of any collected arc is at blue level at most $4k+4\ell \leq 5\ell$ and at red level at most k.

A significant technical difference between the two collection processes is that, in the new process, we do not have the unique charging property that was utilized in the preceding analysis. Nevertheless, we do have a weaker property that we detail next.

Suppose that we have collected 2ℓ ℓ-deep blue arcs, as just described. See Figure 12. We thus obtain $\Theta(\ell^2)$ vertices along these arcs, all contained in W_0 and lying at combined level $4k+5\ell \leq 6\ell$. We claim that each such vertex q is collected in this fashion by at most $O(k^2)$ vertices v.

Consider such a vertex q, and attempt to trace back from q to determine the charging vertex v as follows. Proceed from q along the ℓ-deep blue arc $\delta_{\beta'}$ that contains q, until the first time we reach a vertex q' at red level $\leq k$. (This will happen either when we reach $\bar{\delta}_r^{(3)}$ or earlier.) The red surface incident to any charging vertex v must be one of the $\leq k$ red graphs that lie below q'. (Clearly, $\tilde{f}_r$ is one of these graphs.) Pick any of these graphs, $\tilde{f}_{r'}$, and continue to trace $\delta_{\beta'}$ backward until the

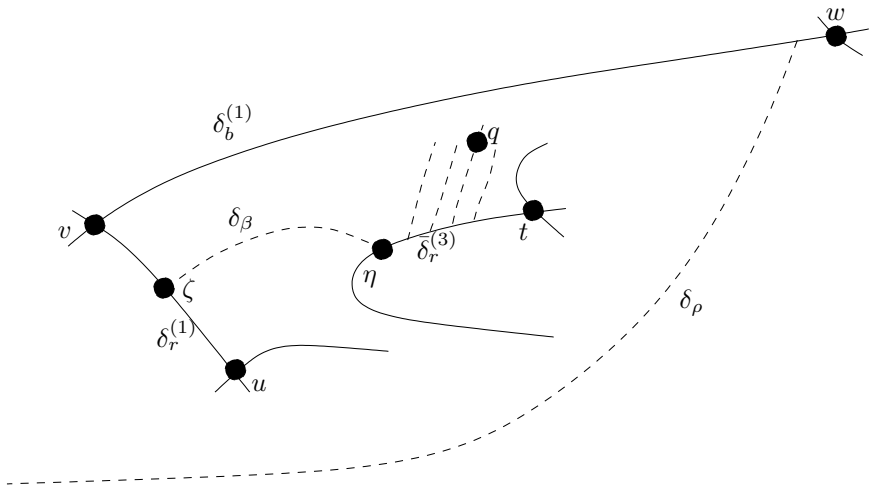


FIG. 12. *The case in which w lies to the right of t and at a high level. The figure depicts the subcase in which there are many deep blue arcs emanating from $\bar{\delta}_r^{(3)}$.*

first time it actually intersects $\tilde{f}_{r'}$. If the stopping point is at red level $> k$, then r' is a wrong guess. We thus keep picking candidate graphs in this fashion until, for one of them, $\tilde{f}_{r''}$, the backward tracing of $\delta_{\beta'}$ reaches $\tilde{f}_{r''}$ at red level $\leq k$. Once this situation is attained, we trace the red curve $\gamma_{r''}$ that we have hit, in the negative x-direction, until the first time we reach a point ν whose blue level is at most $4k$. (As above, if no such point exists, then r'' is a wrong guess, and we keep trying with different candidates $\tilde{f}_{r'''}$.) The blue arc incident to a charging vertex v that is incident to $\tilde{f}_{r''}$ must then correspond to one of the $\leq 4k$ blue graphs lying below ν. ($\tilde{f}_b$ is clearly one of them when $r'' = r$.) Now note that knowing which red and blue arcs are incident to v determines v uniquely. We have thus shown that there are only $O(k^2)$ possible vertices v that can charge q. Hence, using [7], the number of vertices v in this subcase is

$$O(k^2) \cdot O\left(\frac{1}{\ell^2} V_{\leq 6\ell}(n)\right) = O\left(k^2 \ell^2 V_0\left(\frac{n}{6\ell}\right)\right).$$

Similarly, if we collect an ℓ-terminal blue arc, the terminal event along it is charged by at most $O(k^2)$ vertices v, and there are $O(\ell^2 n \lambda_5(n))$ such events. The number of vertices v in this subcase is thus $O(k^2 \ell^2 n \lambda_5(n))$.

We are left to treat the case in which we have collected 2ℓ ℓ-shallow blue arcs. Note that their starting points on $\bar{\delta}_r^{(3)}$ are at combined level at most $5k + 4\ell \leq 5\ell$. Trace any such arc $\delta_{\beta'}$ to its end-point η', which lies on some red edge of ∂W_0, and at combined level $\leq 6\ell$. Several cases can arise, as depicted in Figure 13.

(a) $\eta' \in \delta_r^{(1)}$, and we make a *right* turn from δ_β to $\delta_r^{(3)}$ at η: See Figure 13(a). In this case, we trace $\delta_r^{(3)}$ from η to the left (in the negative x-direction). Since η lies to the left of w, it is easily seen that this tracing of $\delta_r^{(3)}$ must reach a local x-minimum that lies to the right of v. Such cases, however, were ruled out above, where the number of vertices v for which a terminal event on $\delta_r^{(3)}$ can be reached in this fashion was bounded by $O(k^3 n \lambda_5(n))$ in (1).

(a') $\eta' \in \delta_r^{(1)}$, and we make a *left* turn from δ_β to $\delta_r^{(3)}$ at η: See Figure 13(a'). This

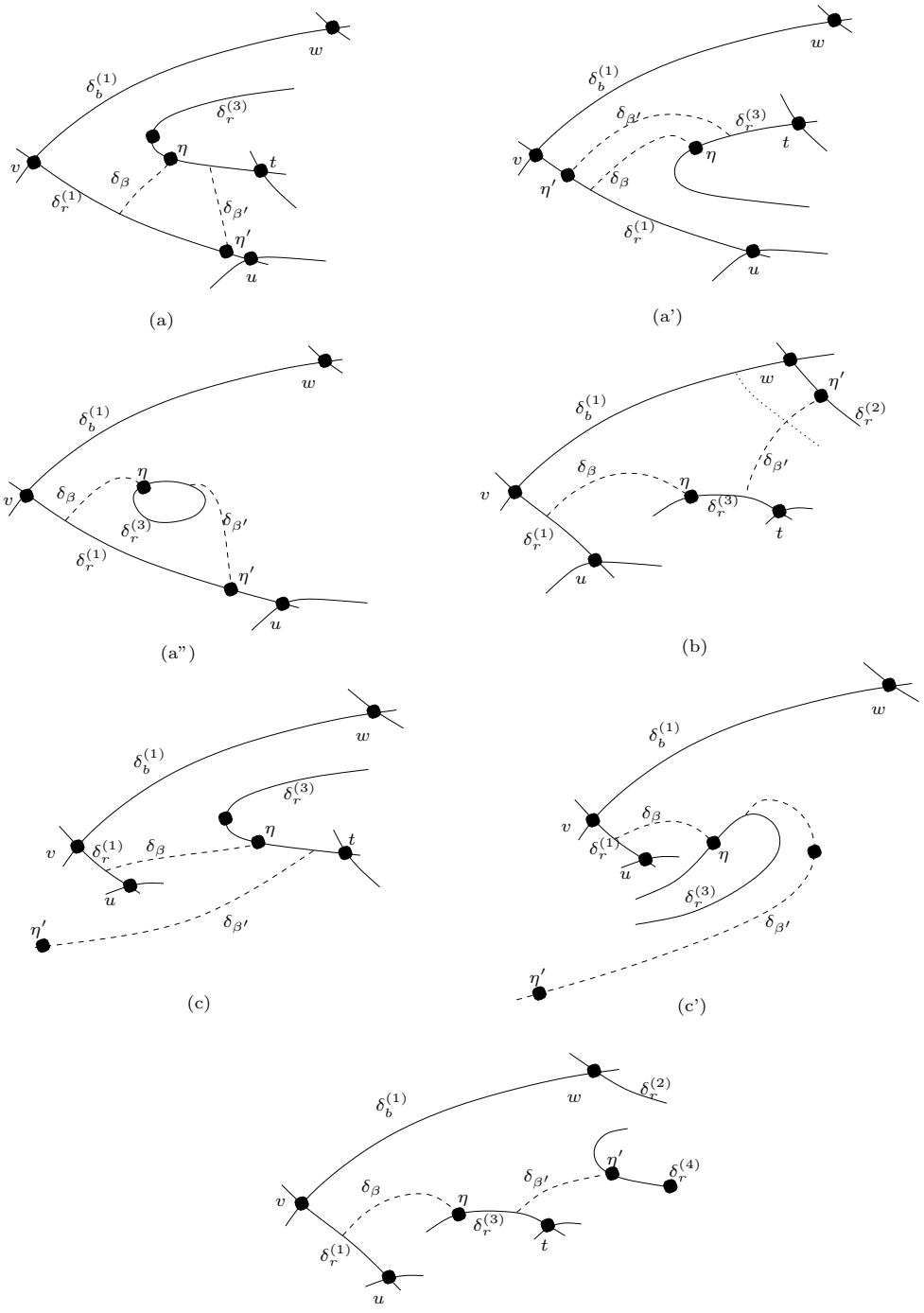


FIG. 13. *Various cases of ℓ-shallow blue arcs that emanate from $\bar{\delta}_r^{(3)}$.*

case can arise only when $\delta_{\beta'}$ lands back on $\delta_r^{(1)}$, between v and δ_β, as in Figure 13(a'). (Otherwise, the component of $\tau_{r,g,y}$ that contains $\delta_r^{(3)}$ would have been forced to be bounded: It has to be contained in the region bounded by δ_β, $\delta_{\beta'}$, $\delta_r^{(3)}$, and $\delta_r^{(1)}$; see Figure 13(a"). This possibility, however, has already been ruled out, as in case (a) above.) Observe that there are overall at most $4k$ arcs $\delta_{\beta'}$ that land back on $\delta_r^{(1)}$ in this fashion. Therefore, at least $2\ell - 4k$ of the ℓ-shallow arcs emanating from $\bar{\delta}_r^{(3)}$ do not belong to case (a').

(b) $\eta' \in \delta_r^{(2)}$: See Figure 13(b). This case can arise only when we make a left turn from δ_β to $\delta_r^{(3)}$ at η, or else $\delta_r^{(3)}$ would have to be a bounded component, as in case (a'). (The configuration would have looked like a "mirror image" of the one depicted in Figure 13(a").) Any red k-shallow arc that emanates from $\delta_b^{(1)}$ must then cross either δ_β or $\delta_{\beta'}$. At most k of these red arcs can cross δ_β, so at least k of them cross each ℓ-shallow arc $\delta_{\beta'}$ that falls into case (b). Since any of these k-shallow red arcs can cross only k blue arcs, it follows that at most k of the ℓ-shallow arcs emanating from $\bar{\delta}_r^{(3)}$ belong to case (b). Since only at most $4k$ arcs fall into case (a'), we conclude that one of the cases (a), (c), or (d) must arise for at least $2\ell - 5k > \ell$ arcs $\delta_{\beta'}$.

(c) η' lies to the left of v. This case cannot arise when we make a right turn from δ_β to $\delta_r^{(3)}$ at η (see Figure 13(c)), for then we could reach, in the opposite direction, a local x-extremum on $\delta_r^{(3)}$, as in case (a). However, if we make a left turn at η, as shown in Figure 13(c'), then $\delta_{\beta'}$ must leave $\delta_r^{(3)}$ in the positive x-direction, or else it would have been "trapped" between $\delta_b^{(1)}$ on one side and $\delta_r^{(1)}$, δ_β, and $\delta_r^{(3)}$ on the other side, which would make it impossible for $\delta_{\beta'}$ to reach to the left of v. Hence $\delta_{\beta'}$ must have a locally x-maximal point before it reaches η'; since this is a terminal event, this contradicts the shallowness of $\delta_{\beta'}$.

(d) η' lies to the right of v on a new edge $\delta_r^{(4)}$ of ∂W_0, different from $\delta_r^{(i)}$, for $i = 1, 2, 3$: See Figure 13(d). In this case, we trace $\delta_r^{(4)}$ from η' in the positive x-direction, and we will not reach any vertex of W_0. (We have already exhausted all such vertices to the right of v.) The tracing will thus reach a terminal event. Since η' lies at combined level at most $5k + 5\ell \leq 6\ell$, this also bounds the 3-colored level of the terminal event. Hence, arguing as above, the number of such events is $O(\ell^2 n \lambda_5(n))$, and each of them is charged by only $O(\ell)$ vertices v. To see the latter claim, we spell out, for the sake of completeness, a modified version of a previous argument.

Trace $\delta_r^{(4)}$ back (in either direction, if more than one direction is applicable, as there may be two choices for r and for $\delta_r^{(4)}$) from the terminal event until the first time we reach a point whose combined level (including the blue level) is at most 6ℓ. (This will be either at η' or earlier.) Then any charging vertex v is a vertex incident to $\tilde{f}_r$, $\tilde{f}_g$, $\tilde{f}_y$, and to some $\tilde{f}_b$ from the at most 6ℓ blue surfaces that lie below the stopping point. In other words, the terminal event can be charged by at most $O(\ell)$ vertices v, implying that the number of vertices v that fall into this final subcase is $O(\ell^3 n \lambda_5(n))$.

This completes the consideration of all possible situations that arise with vertices v of index at most 3. Collecting all of the bounds obtained during the analysis of such vertices leads to the following equation:

$$V_0^{(3)}(n) = O\left(\left(k^2\ell^2 + \ell^3\right) n\lambda_5(n) V_0^{(2)}(n) + k^4 V_0^{(2)}\left(\frac{n}{4k}\right) \right.$$
$$\left. + k^2 V_0^{(3)}\left(\frac{n}{5k}\right) + \ell^4 V_0^{(2)}\left(\frac{n}{5\ell}\right) + k^2\ell^2 V_0^{(3)}\left(\frac{n}{6\ell}\right)\right),$$

which proves the lemma. □

3.3. Putting it all together. Recall that in section 3.2 we handled only regular vertices v. To complete the counting, we have to add the number of irregular vertices to each of the above bounds on the quantities $V_0^{(j)}(n)$. Since there are only $O(n\lambda_5(n))$ irregular vertices, this does not affect any of these asymptotic estimates. Thus, collecting the bounds obtained in Lemmas 3.3–3.5, we obtain the following recurrence relations:

$$V_0^{(0)}(n) = O(n\lambda_5(n)),$$

$$V_0^{(1)}(n) = O(n\lambda_5(n)),$$

$$V_0^{(2)}(n) = O\left(k^3 n\lambda_5(n) + k^2 V_0^{(3)}\left(\frac{n}{5k}\right)\right),$$

$$V_0^{(3)}(n) = O\left(\left(k^2\ell^2 + \ell^3\right) n\lambda_5(n) + V_0^{(2)}(n) + k^4 V_0^{(2)}\left(\frac{n}{4k}\right)\right.$$

$$\left. + k^2 V_0^{(3)}\left(\frac{n}{5k}\right) + \ell^4 V_0^{(2)}\left(\frac{n}{5\ell}\right) + k^2\ell^2 V_0^{(3)}\left(\frac{n}{6\ell}\right)\right).$$

We choose different values of k in the recurrences for $V_0^{(2)}$ and for $V_0^{(3)}$ and denote them by k_2 and k_3, respectively. These values, together with ℓ, are chosen to be sufficiently large constants satisfying $\ell = k_3^{1/(c\varepsilon)}$ and $k_2 = \ell^{1/(c\varepsilon)}$ for an arbitrarily small but prescribed positive constant ε and for some fixed small positive fraction c. (Note that this choice of parameters satisfies $\ell \gg k_3$, which was needed in our analysis.) We also require that k_3^{ε} be sufficiently large. The recurrence for $V_0^{(3)}$ then becomes

$$\begin{aligned}
V_0^{(3)}(n) &= O\left(\left(k_2^{c\varepsilon(2+2c\varepsilon)} + k_2^{3c\varepsilon}\right) n\lambda_5(n) + V_0^{(2)}(n) + k_2^{4c^2\varepsilon^2} V_0^{(2)}\left(\frac{n}{4k_2^{c^2\varepsilon^2}}\right)\right.\\
&\qquad \left. + k_2^{2c^2\varepsilon^2} V_0^{(3)}\left(\frac{n}{5k_2^{c^2\varepsilon^2}}\right) + k_2^{4c\varepsilon} V_0^{(2)}\left(\frac{n}{5k_2^{c\varepsilon}}\right) + k_2^{2c\varepsilon+2c^2\varepsilon^2} V_0^{(3)}\left(\frac{n}{6k_2^{c\varepsilon}}\right)\right)\\
&= O\left(k_2^{3+2c\varepsilon} n\lambda_5(n) + k_2^2 V_0^{(3)}\left(\frac{n}{5k_2}\right) + k_2^{2+4c^2\varepsilon^2} V_0^{(3)}\left(\frac{n}{20k_2^{1+c^2\varepsilon^2}}\right)\right.\\
&\qquad + k_2^{2c^2\varepsilon^2} V_0^{(3)}\left(\frac{n}{5k_2^{c^2\varepsilon^2}}\right) + k_2^{2+4c\varepsilon} V_0^{(3)}\left(\frac{n}{25k_2^{1+c\varepsilon}}\right)\\
&\qquad \left. + k_2^{2c\varepsilon(1+c\varepsilon)} V_0^{(3)}\left(\frac{n}{6k_2^{c\varepsilon}}\right)\right).
\end{aligned}$$

As in other works where similar recurrences have been derived (see, e.g., [23]), it is easy to show, using induction on n, that, with an appropriate choice of c and k_2 (where the choice of k_2 depends on ε but the choice of c does not), the solution of this recurrence is

$$V_0(n) = V_0^{(3)}(n) = O(n^{2+\varepsilon})$$

for any ε, where the constant of proportionality depends on ε. We have thus shown the following.

THEOREM 3.6. *The complexity of the Euclidean Voronoi diagram of a set of n lines in $\mathbb{R}^3$ with four distinct orientations is $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.*

Remark 1. Inspecting the proof of Theorem 3.6, we see that it is fairly general and does not explicitly use the fact that the sites are lines. It can thus be extended to the case of the Voronoi diagram of any reasonable collection of sites (of constant description complexity), which is the union of four subfamilies, under any reasonable metric in $\mathbb{R}^3$, provided that (i) we have a near-quadratic bound for the complexity of the diagram of any three of the given families and (ii) any four sites determine at most eight Voronoi vertices. We strongly suspect that the requirement (ii) can be dropped. This would require us to handle vertices v that have index $x \geq 4$, which in turn would have made the preceding analysis more complicated, mainly by having to use additional thresholds for shallowness (like the k and ℓ that we used). Still, it seems plausible that the analysis could go through.

4. More than four orientations. The case of an arbitrary number c of orientations is easy to handle by noting that any vertex v of the full Voronoi diagram $Vor(L)$ is also a vertex of the diagram of the set of all lines whose orientations are equal to the (at most) four orientations of the lines that are (equally) nearest to v. Let $u_1, \ldots, u_c$ denote the given orientations. Let L_j, for $j = 1, \ldots, c$, denote the set of lines in L at orientation u_j, and put $n_j = |L_j|$. Then $\sum_{j=1}^{c} n_j = n$. Suppose, without loss of generality, that $n_1 \leq n_2 \leq \cdots \leq n_c$. The number of vertices of $Vor(L)$ is at most $\sum_{i<j<k<l} V_{ijkl}$, where V_{ijkl} is the number of vertices of $Vor(L_i \cup L_j \cup L_k \cup L_l)$. By Theorem 3.6, $V_{ijkl} = O((n_i + n_j + n_k + n_l)^{2+\varepsilon}) = O(n_l^{2+\varepsilon})$. Hence the complexity of $Vor(L)$ is at most $O(\sum_{i<j<k<l} n_l^{2+\varepsilon}) = O(\sum_{l=4}^{c} l^3 n_l^{2+\varepsilon})$. As is easily verified, the maximum value of this latter sum is $O(c^3 n^{2+\varepsilon})$. We thus obtain the following theorem, the main result of the paper.

THEOREM 4.1. *The combinatorial complexity of the Euclidean Voronoi diagram of n lines in three dimensions, where the lines have $1 \leq c \leq n$ distinct orientations, is $O(c^3 n^{2+\varepsilon})$ for any $\varepsilon > 0$.*

COROLLARY 4.2. *The combinatorial complexity of the Euclidean Voronoi diagram of n lines in $\mathbb{R}^3$ that have a constant number of distinct orientations is $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.*

Remark 2. As shown in [22], the complexity of the diagram in the general case, without any restrictions on the orientations of the lines (that is, when $c = n$), is $O(n^{3+\varepsilon})$. This leads us to conjecture that the bound in Theorem 4.1 can be improved to at least $O(cn^{2+\varepsilon})$ for any $\varepsilon > 0$. The latter bound is consistent with the result of [22] (when $c = O(n)$) and with Corollary 4.2 (when $c = O(1)$) and might be easier to obtain than a near-quadratic bound like $O(n^{2+\varepsilon})$ for any $1 \leq c \leq n$. (Nevertheless, in line with the general conjecture concerning 3-dimensional Voronoi diagrams, we conjecture that the latter bound does indeed hold independently of c.)

Appendix. In this appendix, we provide a study of the geometric structure of *bisectors* and *trisectors*, which are, respectively, the loci of points equidistant from two and three lines. This analysis is useful in its own right, but most of the details are not needed for the main result of this paper.

We begin with the analysis of bisectors, which have also been studied, e.g., in [9]. Consider the bisector H_{ℓ_1,ℓ_2} between two lines ℓ_1, ℓ_2 at different orientations. Without loss of generality, by translating, rotating, and scaling 3-space, we may as-

sume that ℓ_1 and ℓ_2 are both horizontal, and ℓ_1 (resp., ℓ_2) passes through $(0,0,1)$ (resp., $(0,0,-1)$) and forms a horizontal angle of α (resp., $-\alpha$) with the positive x-direction for $\alpha \in (-\pi/2, \pi/2]$.

The squared distance of a point (x,y,z) from ℓ_1 is

$$d^2((x,y,z),\ell_1) = x^2 + y^2 + (z-1)^2 - (x\cos\alpha + y\sin\alpha)^2,$$

and the squared distance of (x,y,z) from ℓ_2 is

$$d^2((x,y,z),\ell_2) = x^2 + y^2 + (z+1)^2 - (x\cos\alpha - y\sin\alpha)^2.$$

Hence the equation of H_{ℓ_1,ℓ_2} is

$$x^2 + y^2 + (z-1)^2 - (x\cos\alpha + y\sin\alpha)^2 = x^2 + y^2 + (z+1)^2 - (x\cos\alpha - y\sin\alpha)^2,$$

or

$$z = -xy\sin\alpha\cos\alpha.$$

This is the equation of a hyperbolic paraboloid. It has two sets of generating lines, one set consisting of lines parallel to the xz-plane and the other consisting of lines parallel to the yz-plane. Specifically, lines in the first family have the following form, parametrized over $t \in \mathbb{R}$:

$$\lambda_t : \qquad y = -\frac{t}{\sin\alpha\cos\alpha}, \qquad z = tx.$$

Similarly, lines in the second family have the form, parametrized over $s \in \mathbb{R}$,

$$\bar{\lambda}_s : \qquad x = -\frac{s}{\sin\alpha\cos\alpha}, \qquad z = sy.$$

We can project H_{ℓ_1,ℓ_2} onto the xy-plane π_0 bijectively and note that the generating lines project to lines parallel to the axes.

Fix a line λ_t of the first family, having parameter t. Let ℓ_3 be a differently oriented line passing through some point $\mathbf{a} = (a_1, a_2, a_3)$ and having direction $\mathbf{u} = (u_1, u_2, u_3)$, which is a unit vector along ℓ_3 and is common to all input lines of a fixed color. By our general position assumption, we may assume that $u_3 \neq 0$, i.e., that the direction $\mathbf{u}$ is not coplanar with the directions of l_1 and l_2. Without loss of generality, we assume that $\mathbf{a}\cdot\mathbf{u} = 0$. The distance between a point $\mathbf{w} = \mathbf{w}(x)$ on λ_t, parametrized as $(x, -t/(\sin\alpha\cos\alpha), tx)$, and ℓ_3 is

$$d^2(\mathbf{w},\ell_3) = \|\mathbf{w}-\mathbf{a}\|^2 - ((\mathbf{w}-\mathbf{a})\cdot\mathbf{u})^2 = \|\mathbf{w}-\mathbf{a}\|^2 - (\mathbf{w}\cdot\mathbf{u})^2$$

$$= (x-a_1)^2 + \left(\frac{t}{\sin\alpha\cos\alpha} + a_2\right)^2 + (tx-a_3)^2 - \left(xu_1 - \frac{tu_2}{\sin\alpha\cos\alpha} + txu_3\right)^2.$$

Consider the function

$$F(x) = d^2(\mathbf{w}(x),\ell_3) - d^2(\mathbf{w}(x),\ell_1)$$

$$= (x-a_1)^2 + \left(\frac{t}{\sin\alpha\cos\alpha} + a_2\right)^2 + (tx-a_3)^2 - \left(xu_1 - \frac{tu_2}{\sin\alpha\cos\alpha} + txu_3\right)^2$$

$$- x^2 - \frac{t^2}{\sin^2\alpha\cos^2\alpha} - (tx-1)^2 + \left(x\cos\alpha - \frac{t}{\cos\alpha}\right)^2.$$

Note that $F(x)$ is positive (resp., zero, negative) if the ball centered at $\mathbf{w}(x)$ and touching ℓ_1 and ℓ_2 is disjoint from (resp., touches, intersects) ℓ_3. Hence the locus of the roots of $F(x)$, as we trace them by varying t from $-\infty$ to $+\infty$, is the *trisector* $\tau_{\ell_1,\ell_2,\ell_3}$—the locus of all centers of balls that touch ℓ_1, ℓ_2, ℓ_3 simultaneously.

The function $F(x)$ is quadratic (for any fixed t), and its global behavior along λ_t depends largely on the sign of the coefficient of x^2, which is

$$A(t) = 1 + t^2 - (u_1 + tu_3)^2 - 1 - t^2 + \cos^2\alpha = \cos^2\alpha - (u_1 + tu_3)^2.$$

Hence, if $A(t) > 0$, then $F(x)$ is convex and is positive at $x = \pm\infty$, meaning that at the extremities of λ_t the ball touching ℓ_1, ℓ_2 is disjoint from ℓ_3 (we are in the *free region* associated with ℓ_3), whereas if $A(t) < 0$, then $F(x)$ is concave, and at the extremities of λ_t we are in the *intersection region* of ℓ_3.

In other words, assuming, as above, that $u_3 \neq 0$, and, for specificity, that $u_3 > 0$, we have that $A(t) < 0$ if and only if $|u_1 + tu_3| > \cos\alpha$ or

$$t > \frac{-u_1 + \cos\alpha}{u_3} \quad \text{or} \quad t < \frac{-u_1 - \cos\alpha}{u_3}.$$

The corresponding critical y-values are

$$y_T = \frac{u_1 + \cos\alpha}{u_3 \sin\alpha\cos\alpha} \quad \text{and} \quad y_B = \frac{u_1 - \cos\alpha}{u_3 \sin\alpha\cos\alpha},$$

and we denote the corresponding horizontal *critical lines* by $\lambda^{(T)}$ and $\lambda^{(B)}$, respectively. (Note that the critical lines depend *only* on the orientation $\mathbf{u}$ of l_3.)

We next apply a symmetric analysis to lines in the other family. We obtain that the critical x-values where the corresponding quadratic function changes from being convex to being concave are

$$x_R = \frac{u_2 + \sin\alpha}{u_3 \sin\alpha\cos\alpha} \quad \text{and} \quad x_L = \frac{u_2 - \sin\alpha}{u_3 \sin\alpha\cos\alpha};$$

the corresponding vertical critical lines are denoted by $\bar{\lambda}^{(R)}$ and $\bar{\lambda}^{(L)}$.

We next claim that, for $|t|$ sufficiently large, the line λ_t intersects the trisector in exactly two points. For this, we need to show that the discriminant of the quadratic equation $F(x)$ becomes positive as $|t|$ tends to ∞.

Write $F(x)$ as $A(t)x^2 + 2B(t)x + C(t)$, where

$$\begin{aligned} A(t) &= \cos^2\alpha - (u_1 + tu_3)^2, \\ B(t) &= -a_1 - a_3 t + \tfrac{u_2 t(u_1 + tu_3)}{\sin\alpha\cos\alpha}, \\ C(t) &= a_1^2 + \left(\tfrac{t}{\sin\alpha\cos\alpha} + a_2\right)^2 + a_3^2 - \tfrac{t^2 u_2^2}{\sin^2\alpha\cos^2\alpha} - \tfrac{t^2}{\sin^2\alpha\cos^2\alpha} - 1 + \tfrac{t^2}{\cos^2\alpha}. \end{aligned}$$

As $|t|$ tends to ∞, the sign of the discriminant $\Delta(t)$ depends only on the coefficients of t^2 in these three expressions. That is, the limit of Δ/t^4 is

$$\lim_{|t|\to\infty} \frac{B^2(t) - A(t)C(t)}{t^4} = \lim_{|t|\to\infty} \left[\left(\frac{B(t)}{t^2}\right)^2 - \frac{A(t)}{t^2} \cdot \frac{C(t)}{t^2} \right]$$

$$= \frac{u_2^2 u_3^2}{\sin^2\alpha\cos^2\alpha} + u_3^2 \cdot \left(\frac{1}{\cos^2\alpha} - \frac{u_2^2}{\sin^2\alpha\cos^2\alpha} \right) = \frac{u_3^2}{\cos^2\alpha} > 0.$$

That is, for large values of $|t|$, the trisector $\tau_{\ell_1,\ell_2,\ell_3}$ meets λ_t at two points $w_1(t)$, $w_2(t)$. The asymptotic values of these roots are

$$\lim_{|t|\to\infty} w_{1,2}(t) = \lim_{|t|\to\infty} \frac{-B(t) \pm \sqrt{\Delta(t)}}{A(t)} = \lim_{|t|\to\infty} \frac{-B(t)/t^2 \pm \sqrt{\Delta(t)/t^4}}{A(t)/t^2}$$

$$= \frac{-\frac{u_2 u_3}{\sin\alpha\cos\alpha} \pm \frac{u_3}{\cos\alpha}}{-u_3^2} = \frac{u_2 \pm \sin\alpha}{u_3 \sin\alpha\cos\alpha}.$$

That is, $w_1(t)$ and $w_2(t)$ tend to x_L and x_R, respectively.

Symmetrically, there always exist two intersection points of $\tau_{\ell_1,\ell_2,\ell_3}$ with the lines $\bar{\lambda}_s$, as $|s|$ tends to ∞, and their limits are at the ordinates y_B and y_T.

We have thus shown that any sufficiently large circle intersects the trisector at eight points. We denote the points "at infinity" that lie on the vertical critical lines $\bar{\lambda}^{(L)}$, $\bar{\lambda}^{(R)}$ as $v_{LB}, v_{LT}, v_{RB}, v_{RT}$, where v_{LB} (resp., v_{LT}) is the bottom (resp., top) end of $\bar{\lambda}^{(L)}$, and similarly for the other two points. The points at infinity on the horizontal lines are denoted, in a similar manner, as $h_{LB}, h_{LT}, h_{RB}, h_{RT}$. See Figure 14(a) for an illustration.

Assuming that the trisector is nonsingular, it has exactly four unbounded components, each connecting two of these points at infinity. We next proceed to classify the structure of these components.

The function $F(x)$ becomes linear along each of the horizontal critical lines $\lambda^{(T)}$, $\lambda^{(B)}$, and thus each of these two critical lines is intersected by the trisector exactly once; symmetrically, this also holds for $\bar{\lambda}^{(L)}$, $\bar{\lambda}^{(R)}$. Number the eight points at infinity in a cyclic order. Then it is clear that each odd-numbered point must be connected to an even-numbered point, since the components of the trisector are disjoint. Hence, v_{LT} can be connected to h_{LT}, v_{LB}, h_{RB}, or to v_{RT}, and similarly for the other points at infinity.

Consider the second case, in which v_{LT} is connected to v_{LB} via one component γ_1 of the trisector. This component crosses the two critical horizontal lines $\lambda^{(B)}$, $\lambda^{(T)}$ (each exactly once). In this case, no other component of the trisector can intersect any of these lines, so each of the remaining three components is fully contained in one of the three horizontal slabs delimited by $\lambda^{(B)}$ and $\lambda^{(T)}$, and each of these slabs contains exactly one such component. It then follows that these components must connect h_{LT} to h_{LB}, v_{RT} to h_{RT}, and v_{RB} to h_{RB}. Moreover, γ_1 must cross $\bar{\lambda}^{(L)}$ (exactly once), and one of the two components on the right must cross $\bar{\lambda}^{(R)}$ (exactly once). Hence the trisector has a shape similar to that shown in Figure 14(b).

Consider next the third case, in which v_{LT} is connected to h_{RB} via one component γ_1 of the trisector. Another component, γ_2, must connect v_{RT} to h_{RT}. We have two subcases.

In the first subcase, h_{LT} is connected to v_{RB}, and h_{LB} is connected to v_{LB}. In this case, none of the components can cross any of its asymptotes. See Figure 14(a).

In the second subcase, h_{LT} is connected to h_{LB}, and v_{LB} is connected to v_{RB}. In this case, we must allow each of the lines $\lambda^{(B)}$, $\bar{\lambda}^{(L)}$ to be crossed (once) by some component. See Figure 14(c). This figure depicts one of several possible subcases, depending on which component crosses which critical line. In Figure 14(c) the component connecting v_{LT} to h_{RB} crosses all four critical lines, but it might also be possible for this component to cross only $\lambda^{(T)}$ and $\bar{\lambda}^{(R)}$ or to cross just one more critical line and let the left and/or bottom components cross the other one or two critical lines (in a manner similar to that of the top-right component in Figure 14(b)).

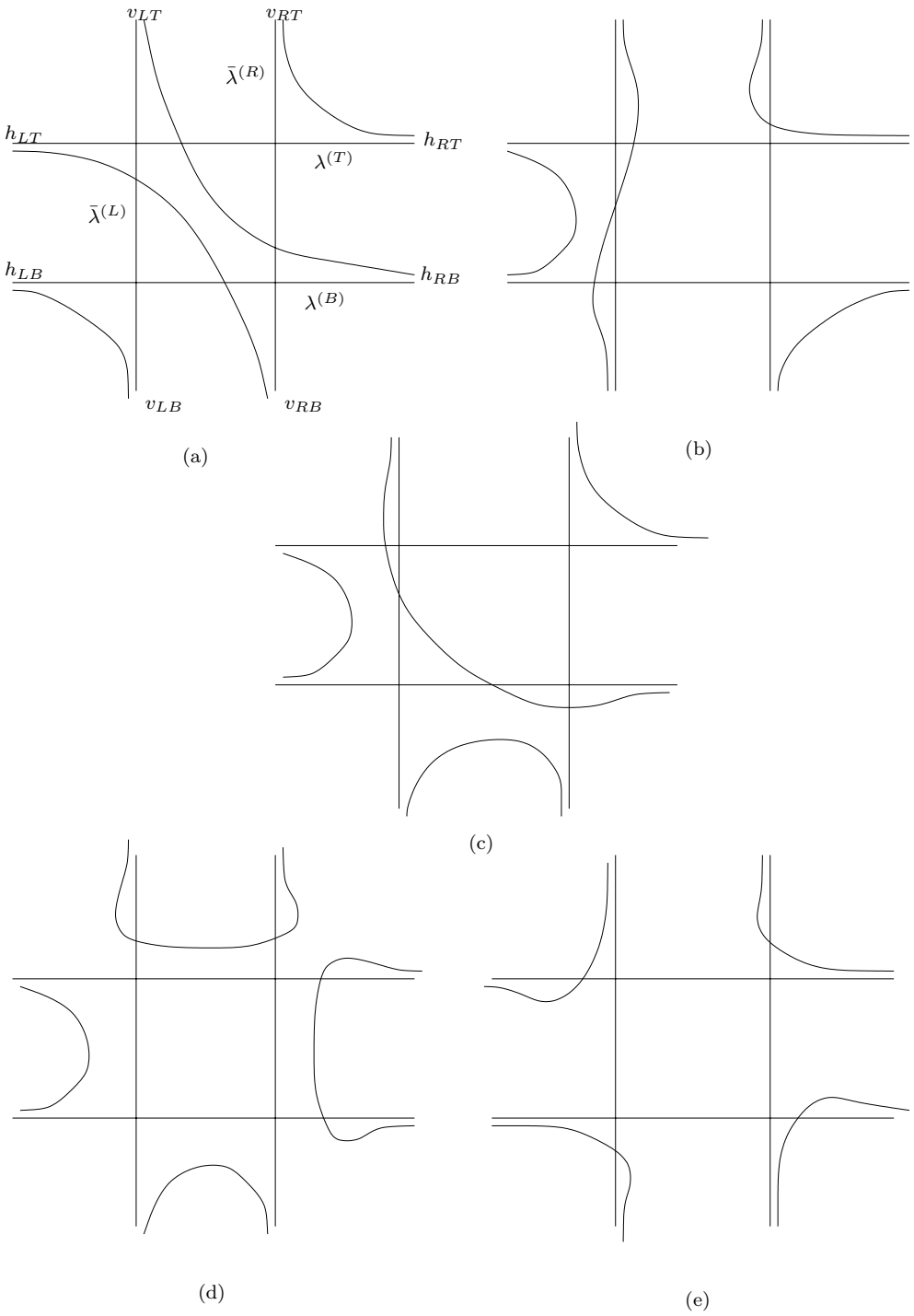


FIG. 14. *The various possible structures of a trisector.*

If none of the above cases occur, including their various symmetric variants, then each end of each critical line must be connected to one of its two neighbors in the above cyclic order. Only two cases are possible.

In the first subcase, h_{LT} is connected to h_{LB}, v_{LB} is connected to v_{RB}, h_{RT} is connected to h_{RB}, and v_{LT} is connected to v_{RT}. As above, we must let some of these components cross some of their asymptotes to ensure that each of the four critical lines is crossed once by the trisector. See Figure 14(d), which, as above, depicts just one of several possible subcases.

In the second subcase, h_{LT} is connected to v_{LT}, h_{LB} is connected to v_{LB}, h_{RT} is connected to v_{RT}, and h_{RB} is connected to v_{RB}. Again, we must let some of these components cross some of their asymptotes. One of several possible such configurations is shown in Figure 14(e).

We also note that each trisector is an algebraic curve of degree 4. By Harnack's theorem [13], the number of components of a real nonsingular algebraic plane curve of degree d is at most $(d-1)(d-2)/2+1$. Hence the number of components of each trisector is at most $3 \cdot 2/2 + 1 = 4$. Since it has exactly four unbounded components, we conclude that these are *all* the components of the trisector. In particular, no component of any trisector is bounded. This completes the classification of the trisectors.

Remark 3. We conjecture that, up to symmetry, only trisectors of the kind shown in Figure 14(b) are possible. A program that we have written to explore the structure of trisectors has revealed only trisectors of this kind, after several tens of thousands of tests with randomly generated lines.

Acknowledgments. We are grateful to Mark de Berg, who raised the possibility of improving the bound $O(c^4 n^{2+\varepsilon})$, which appeared in preliminary versions of this paper, to the bound $O(c^3 n^{2+\varepsilon})$ that is currently proved in section 4. We are also grateful to Emo Welzl and the European Graduate Program on Combinatorics, Geometry and Computation for hosting us at ETH Zürich, where the work on this paper was initiated.

REFERENCES

[1] P. K. Agarwal and M. Sharir, *Pipes, cigars, and kreplach: The union of Minkowski sums in three dimensions*, Discrete Comput. Geom., 24 (2000), pp. 645–685.

[2] B. Aronov, *A lower bound on Voronoi diagram complexity*, Inform. Process. Lett., 83 (2002), pp. 183–185.

[3] F. Aurenhammer, *Power diagrams: Properties, algorithms and applications*, SIAM J. Comput., 16 (1987), pp. 78–96.

[4] F. Aurenhammer and R. Klein, *Voronoi diagrams*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., North–Holland, Amsterdam, 2000, pp. 201–290.

[5] J.-D. Boissonnat, M. Sharir, B. Tagansky, and M. Yvinec, *Voronoi diagrams in higher dimensions under certain polyhedral distance functions*, Discrete Comput. Geom., 19 (1998), pp. 473–484.

[6] L. P. Chew, K. Kedem, M. Sharir, B. Tagansky, and E. Welzl, *Voronoi diagrams of lines in 3-space under polyhedral convex distance functions*, J. Algorithms, 29 (1998), pp. 238–255.

[7] K. L. Clarkson and P. W. Shor, *Applications of random sampling in computational geometry* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[8] H. Edelsbrunner and R. Seidel, *Voronoi diagrams and arrangements*, Discrete Comput. Geom., 1 (1986), pp. 25–44.

[9] G. Elber and M.-S. Kim, *The bisector surface of freeform rational space curves*, in Proceedings of the 13th ACM Symposium on Computational Geometry, ACM, New York, 1997, pp. 473–474.

[10] S. Fortune, *Voronoi diagrams and Delaunay triangulations*, in Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke, eds., CRC Press LLC, Boca Raton, FL, 1997, pp. 377–388.
[11] D. Halperin and M. Sharir, *New bounds for lower envelopes in three dimensions, with applications to visibility in terrains*, Discrete Comput. Geom., 12 (1994), pp. 313–326.
[12] D. Halperin and M. Sharir, *Almost tight upper bounds for the single cell and zone problems in three dimensions*, Discrete Comput. Geom., 14 (1995), pp. 385–410.
[13] A. Harnack, *Uber die Vielfaltigkeit der ebenen algebraischen Kurven*, Math. Ann., 10 (1876), pp. 189–199.
[14] R. Hartshorne, *Algebraic Geometry*, Springer-Verlag, New York, 1977.
[15] C. Icking and L. Ma, *A tight bound for the complexity of Voronoi diagrams under polyhedral convex distance functions in* $3D$, in Proceedings of the 33rd ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 316–321.
[16] V. Klee, *On the complexity of d-dimensional Voronoi diagrams*, Arch. Math. (Basel), 34 (1980), pp. 75–80.
[17] V. Koltun and M. Sharir, *Polyhedral Voronoi diagrams of polyhedra in three dimensions*, in Proceedings of the 18th ACM Symposium on Computational Geometry, ACM, New York, 2002, pp. 227–236.
[18] D. Leven and M. Sharir, *Intersection and proximity problems and Voronoi diagrams*, in Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics, J. T. Schwartz and C.-K. Yap, eds., Lawrence Erlbaum Associates, Hillsdale, NJ, 1987, pp. 187–228.
[19] J. S. B. Mitchell and J. O'Rourke, *Computational geometry column* 42, Internat. J. Comput. Geom. Appl., 11 (2001), pp. 573–582.
[20] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, John Wiley and Sons, New York, 2000.
[21] R. Seidel, *Exact upper bounds for the number of faces in d-dimensional Voronoi diagrams*, in Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 4, P. Gritzman and B. Sturmfels, eds., AMS, Providence, RI, 1991, pp. 517–530.
[22] M. Sharir, *Almost tight upper bounds for lower envelopes in higher dimensions*, Discrete Comput. Geom., 12 (1994), pp. 327–345.
[23] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.
[24] B. Tagansky, *The Complexity of Substructures in Arrangements of Surfaces*, Ph.D. thesis, Tel Aviv University, Tel Aviv, 1996.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 643–653

SETTING PARAMETERS BY EXAMPLE*

DAVID EPPSTEIN†

Abstract. We introduce a class of "inverse parametric optimization" problems, in which one is given both a parametric optimization problem and a desired optimal solution; the task is to determine parameter values that lead to the given solution. We describe algorithms for solving such problems for minimum spanning trees, shortest paths, and other "optimal subgraph" problems and discuss applications in multicast routing, vehicle path planning, resource allocation, and board game programming.

Key words. inverse optimization, parametric search, shortest paths, minimum spanning tree, vehicle routing, adaptive user interfaces, alpha-beta search, evaluation function, randomized algorithms, ellipsoid method

AMS subject classifications. 68Q25, 90C31, 90C35

PII. S0097539700370084

1. Introduction.

Many cars now come equipped with route planning software that suggests a path from the current location to a desired destination. Similar services are also available on the Internet (e.g., from http://maps.yahoo.com/). But although these routes may be found by computing shortest paths in a graph representing the local road system, the "distance" may be a weighted sum of several values other than actual mileage: expected travel time, scenic value, number of turns, tolls, etc. [23]. Different drivers may have different preferences among these values, and they may not be able to clearly articulate these preferences. Can we automatically infer the appropriate weights to use in the sum by observing the routes actually chosen by a driver?

More abstractly, we define an *inverse parametric optimization* problem as follows: we are given as input both a *parametric optimization* problem (that is, a combinatorial optimization problem such as shortest paths, but with the element weights being linear combinations of certain parameters rather than fixed numbers) and a desired optimal solution for the problem.[1] Our task is to determine parameter values such that the given solution is optimal for those values.

Along with the path planning problem described above, one can find many other applications in which one must tune the parameters to an optimization problem:

- In many online services such as web page hosting, data is sent in a star topology from a central server to each user. But in multicast routing of video and other high-bandwidth information, network resources are conserved by sending the data along the edges of a tree, in which some users receive copies of the data from other users rather than from the central server. Natural measures of the quality of each edge in this routing tree include the edge's bandwidth, congestion, delay, packet loss, and possibly monetary charges for

*Received by the editors March 31, 2000; accepted for publication (in revised form) October 16, 2002; published electronically March 5, 2003. This work was supported in part by NSF grant CCR-9912338. A preliminary version of this paper appeared in *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science,* 1999.
http://www.siam.org/journals/sicomp/32-3/37008.html
†Department of Information and Computer Science, University of California, Irvine, CA 92697-3425 (eppstein@ics.uci.edu).
[1]One could more generally allow as input a set of problem-solution pairs, but for most of the problems we consider, any such set can be represented equally well by a single larger problem.

use of that link. Since one can find minimum spanning trees efficiently in the distributed setting [14], it is natural to try to model this routing problem using minimum spanning trees. Given one or more networks with these parameters, and examples of desired routing trees, how can we set the weights of each quality measure so that the desired trees are the minimum spanning trees of their networks?

- Bipartite matching, or the *assignment problem*, is a common formalism for grouping indivisible resources with resource consumers. For instance, the first example given for matching by Ahuja, Magnanti, and Orlin [2] is to assign recently hired workers to jobs, using weights based on such values as aptitude test scores and college grades. One might set the weight of an edge from worker i to job j to be $a_i \cdot p_j$, where a_i is the (known) set of aptitudes of the worker, and p_j is the (unknown) set of parameters describing the combination of aptitudes best fitting the job. Again, it is natural to ask for a way to automatically set the parameters of each job based on experience assigning previously hired workers to those jobs.
- Many board games, such as chess, checkers, or Othello, can be played well by programs based on relatively simple alpha-beta searching algorithms. However, these programs use relatively complex evaluation functions in which the evaluation of a given position can be the sum of hundreds or thousands of terms. Some of these terms may represent the gross material balance of a game (e.g., in chess, one usually normalizes the score so that a pawn is worth 1 point, while a knight may be worth 2.5–3 points) while others represent more subtle features of piece placement, king safety, advanced pawns, etc. The weight of each of these terms may be individually adjusted in order to improve the quality of play. Although there have been some preliminary experiments in using evolutionary learning techniques to tune these weights [25], they are currently usually set by hand. The true test of a game program is in actual play, but programs are also often tuned by using *test suites*, large collections of positions for which the correct move is known. If we are given a test suite, can we automatically set evaluation weights in such a way that a shallow alpha-beta search can find each correct move?

1.1. New results. We show the following theoretical results. For the inverse parametric minimum spanning tree problem, in the case that the number of parameters is a fixed constant, we provide a randomized algorithm with linear expected running time, and a deterministic algorithm with worst-case running time $O(m \log^2 n)$. For the minimum spanning tree, shortest path, matching, and other "optimal subgraph" problems for which the optimization problem can be solved in polynomial time, we show that the inverse optimization problem can also be solved in polynomial time by means of the ellipsoid method from linear programming, even when the number of parameters is large.

In addition, we discuss the problem of setting weights for game evaluation and describe how to solve it within the same inverse parametric optimization framework.

In cases where the initial problem is infeasible (there is no parameter setting for which the desired solution is optimal), our techniques provide a witness for infeasibility: a small number of alternative solutions, one of which must be better than the given solution for any parameter setting. One can then examine these solutions to determine whether the initial solution is suboptimal or whether additional parameters should be added to better model the users' utility functions.

1.2. Relation to previous work. Although there has been considerable work on parametric versions of optimization problems such as minimum spanning trees [1] and shortest paths [26], we are not aware of any prior work in inverting such problems to produce parameter values that match given solutions. One could compute the set of solutions available over the range of parameter values and compare these solutions to the given one, but the number of different solutions would typically grow exponentially with the number of parameters.

The inverse parametric optimization problems considered here are most closely related to *parametric search*, which describes a general class of problems in which one sets the parameters of a parametric problem in order to optimize some criterion. However, in most applications of parametric search, the criterion being optimized is a numeric function of the solution (e.g., the ratio between two linear weights) rather than the solution structure itself. Megiddo [20] describes a very general technique for solving parametric search problems in which one simulates the steps of an optimization algorithm, at each conditional step using the algorithm itself as an oracle to determine which conditional branch to take. However, this technique does not seem to apply to our problems, because the given optimal structure (e.g., a single shortest path) does not give enough information to deduce the conditional branches followed by a shortest path algorithm.

The vehicle routing problem discussed in the introduction was introduced by Rogers and Langley [23]. However, they used a weaker model of optimization (a hill-climbing procedure) and a stronger model of user interaction requiring the user to specify preferences in a sequence of choices between pairs of routes.

The inverse parametric optimization problems we study are also related to previous work on "inverse optimization" problems, in which one is given an optimization problem with a solution and must find new weights for the problem that make the given solution optimal [3, 4, 5, 24, 27, 28]. However, work on these problems has not focused on the parametric aspects of such optimization, which can be interpreted in this framework as requiring the edge weights to be in a linear subspace rather than being unrestricted, and has instead looked at finding a set of weights that is as close as possible to some given set.

2. Minimum spanning trees. In this section, we consider the *inverse parametric minimum spanning tree problem*, in which we are given a fixed tree T in a network in which the weight of each edge e is a linear function $w(e) = \mathbf{c}_e \cdot \mathbf{p}$ (where $\mathbf{p}$ represents the unknown vector of parameter settings and $\mathbf{c}_e$ represents the known value of edge e according to each parameter). Our task is to find a value of $\mathbf{p}$ such that T is the unique minimum spanning tree for the weights $w(e)$.

If we fix a given spanning tree T in a network, a pair of edges (e, f) is defined to be a *swap* if $T \cup \{f\} \setminus \{e\}$ is also a spanning tree; that is, if e is an edge in T, f is not an edge in T, and e belongs to the cycle induced in T by f. T is the unique minimum spanning tree if and only if for every swap (e, f), the weight of f is greater than the weight of e.

Thus we can solve the inverse parametric minimum spanning tree problem as a linear program, in which we have one variable per parameter, an additional variable δ measuring the amount by which the inequality $w(f) > w(e)$ holds, and one constraint $(\mathbf{c}_f - \mathbf{c}_e) \cdot \mathbf{p} \geq \delta$ per swap. To avoid multiple equally good spanning trees we use the maximization of δ as our objective function, and to avoid unboundedness we can add constraints such as $-\mathbf{1} \leq \mathbf{p} \leq \mathbf{1}$. We call these additional constraints *parameter constraints* since they do not depend on the input network.

If the number of variables is a fixed constant, a linear program may be solved in time linear in the number of constraints [21]; however, here the number of constraints may be $\Theta(mn)$. We show how to improve this by a randomized algorithm which takes linear time and by a deterministic algorithm which takes time $O(m \log^2 n)$.

2.1. Randomized spanning tree algorithm. In the conference version of this paper, we described a linear time algorithm for the inverse parametric minimum spanning tree problem, using ideas of Clarkson [9] to bound the sizes of the remaining subproblems after two levels of random sampling. Here we replace this technique by a simpler method of Chan [8].

LEMMA 1 (Chan [8]). *Let P be a class of problems with real-valued solutions such that any instance of size n can be divided in time $T(n)$ into a constant number of subproblems of size αn, $\alpha < 1$, so that the solution of the original instance is the minimum of the subproblem solutions. Further suppose that for any instance of size n and subproblem solution v we can decide whether the solution of another subproblem has a value less than v, in time $T(n)$, and that $T(n) = \Omega(n^\epsilon)$ for some $\epsilon > 0$. Then we can solve any problem in P in time $O(T(n))$.*

Given an instance (G, T) of the inverse parametric spanning tree problem, and given a subset S of the edges, define an instance (G_S, T_S) by contracting all edges in $E(T) \cap S$ and deleting from G all edges in $(E(G) \setminus E(T)) \cap S$. The contraction of T is a tree with edges in $E(T) \setminus S$, and the contraction and deletion of edges in G forms a graph with edges in $E(G) \setminus S$. It is not hard to see that a pair (e, f) forms a swap in (G_S, T_S) if and only if (e, f) is a swap in (G, T) and neither e nor f belongs to S.

THEOREM 2. *We can solve the inverse parametric minimum spanning tree problem for any constant number of parameters in randomized linear expected time.*

Proof. Let the number of parameters be d; then the linear programming formulation of the problem has dimension $d+1$, and its solution is the minimum value of δ determined by some $(d+1)$-tuple of swaps. Partition the edges of G into $2d+3$ equal-cardinality subsets S_i; then at least one of the sets S_i is disjoint from the $2d+2$ swap edges that determine the problem's solution. Therefore, the solution to the problem is the minimum of the solutions to the $2d+3$ subproblems (G_{S_i}, T_{S_i}), each of which has size (number of edges) $(2d+2)/(2d+3)$ times the size of the original instance.

If v is the solution to one of the subproblems, let $\mathbf{p}$ denote the parameter vector associated with v. To test whether another subproblem (G_{S_i}, T_{S_i}) has a smaller solution than v, we use a linear time minimum spanning tree verification algorithm [10, 18] to check whether T_{S_i} is a minimum spanning tree of G_{S_i} for parameter setting $\mathbf{p}$. If not, some linear programming constraint is violated by the solution $\mathbf{p}$ and the subproblem solution is smaller than v.

The time to partition the problem into subproblems and to test subproblem solutions is linear, so by Chan's lemma, the overall time to solve the inverse parametric minimum spanning tree problem is also linear. □

2.2. Deterministic spanning tree algorithm. To solve the inverse parametric minimum spanning tree problem deterministically, we use a different sampling technique of Clarkson [9] and derandomize it using methods very similar to those of an approximate set cover algorithm of Brönnimann and Goodrich [7].

We begin by applying the *multilevel restricted partition* technique of Frederickson [12, 13] to the given tree T.

By introducing dummy edges, we can assume without loss of generality that T is binary and that the root t of T has indegree one. These dummy edges will only

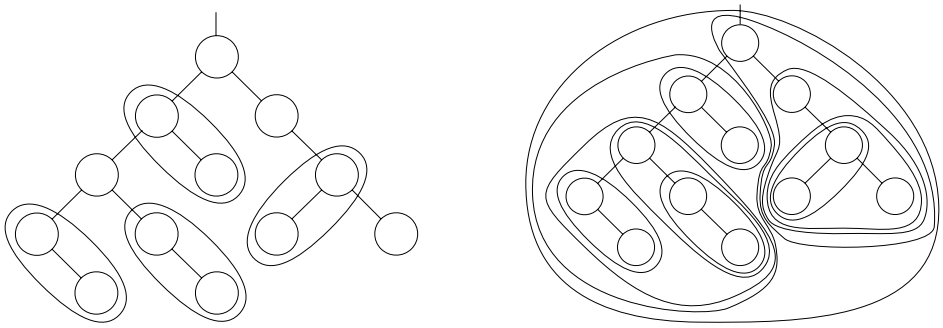


FIG. 1. (a) *Restricted partition of order* 2*;* (b) *multilevel partition.*

be used to form the partition and will not take part in the eventual optimization procedure.

DEFINITION 1. *A restricted partition of order z with respect to a rooted binary tree T is a partition of the vertices of V such that*

1. *each set in the partition contains at most z vertices;*
2. *each set in the partition induces a connected subtree of T;*
3. *for each set S in the partition, if S contains more than one vertex, then there are at most two tree edges having one endpoint in S;*
4. *no two sets can be combined and still satisfy the other conditions.*

Such a partition (for $z = 2$) is depicted in Figure 1(a). In general such a partition can easily be found in linear time by merging sets until we get stuck. Alternatively, by working bottom up we can find an optimal partition in linear time. We will defer until later choosing a value for z; for now we leave it as a free parameter.

LEMMA 3 (Frederickson [13]). *Any order-z partition of a binary tree T has $O(n/z)$ sets in the partition. For $z = 2$ we can find a partition with at most $5n/6$ sets.*

Contracting each set in a restricted partition again gives a binary tree. We form a *multilevel partition* [13] by recursively partitioning this contracted binary tree (Figure 1(b)).

We now use these partitions to construct a set Π of paths in T. For any set S in the multilevel partition, define an *internal edge* of S to be an edge of T that has both endpoints in S, but at most one endpoint in any set in a lower level of the partition, and define an *external edge* of S to be an edge of T that has exactly one endpoint in S. Define a *terminal* of S to be an endpoint of an internal or external edge of S. We form Π by including all paths in T between pairs (u, v), where u and v are both terminals of the same set.

LEMMA 4. *The set of paths Π defined above has the following properties:*

- *There are $O(nz)$ paths in Π.*
- *Each edge in T belongs to $O(z^2 \log_z n)$ paths of Π.*
- *Any path in T can be decomposed into the disjoint union of $O(\log_z n)$ paths of Π.*

Proof. The first property follows immediately from Lemma 3, since each set of the partition contributes $O(z^2)$ paths, there are $O(n/z)$ sets at the bottom level of the partition, and the number of sets decreases at least geometrically at each level. Similarly, the second property follows, since an edge can belong to $O(z^2)$ paths per level and there are $O(\log_z n)$ levels.

Finally, to prove the third property, let p be an arbitrary path in T. We describe a procedure for decomposing p into few paths $\pi_i \in \Pi$. More generally, suppose we have a path p contained in a set S at some level of a multilevel decomposition (note that the whole tree is the set at the highest level of the partition). Then S can be decomposed into at most z sets at the next level of the partition; p has endpoints in at most two of these sets and may pass completely through some other sets. Therefore, p can be decomposed into the union of two smaller paths in the sets containing its endpoints, together with a single path π_i connecting those two sets. By repeating this decomposition recursively at each level of the tree, we obtain a decomposition into at most two paths per level, or $O(\log_z n)$ paths overall. □

We now describe how to use this path decomposition in our inverse optimization problem. For each path $\pi_i \in \Pi$, let A_i denote the set of edges in T belonging to π_i. Let B_i denote the set of nontree edges (u, v) such that π_i is included in the decomposition of the path in T from u to v. The total size of all the sets A_i is $O(nz^2 \log_z n)$ by the second property of Lemma 4, and the total size of all the sets B_i is $O(m \log_z n)$ by the third property of the same lemma. It is not difficult to explicitly construct all sets A_i and B_i in time linear in their total size.

A pair (e, f) is a swap if and only if there is some e for which $e \in A_i$ and $f \in B_i$. With this decomposition, the inverse parametric minimum spanning tree problem becomes equivalent to asking for a parameter $\mathbf{p}$ such that for each i, the weight of every member of A_i is less than the weight of every member of B_i.

For a single value of i, one could solve such a problem by a $(d+1)$-dimensional linear program in which we augment the parameters by an additional variable that is constrained to be greater than each $e \in A_i$ and less than each $f \in B_i$; however, adding a separate variable for each i would make the dimension nonconstant.

Instead, we use a standard derandomization technique from computational geometry, *ϵ-nets*. If we graph the weight of each edge in a d-dimensional space (where again d is one more than the number of parameters), letting the $d-1$ parameter values be independent variables and the weight be the dependent variable, the result is a hyperplane. Suppose that these edges are also given costs, independent of their weights. For any set S of these hyperplanes and any $\epsilon > 0$, define an ϵ-net for vertical line segments to be a subset S' with the following property: if any vertical line segment V intersects a subset of the hyperplanes having cost at least ϵ times the total cost of all hyperplanes, then the same segment must intersect at least one hyperplane in S'. Figure 2 shows an example in which $\epsilon = 1/2$ and all costs are equal. If $1/\epsilon = O(1)$, an ϵ-net of size $O(1)$ can be found in time linear in $|S|$ [6].

Our algorithm can then be described as follows. We will use $\epsilon = 1/4d$.

1. Construct a multilevel partition.
2. Find the set of paths Π connecting terminals of the sets of the partition.
3. Find the sets A_i by listing the edges in each path $\pi \in \Pi$.
4. Find the sets B_i by decomposing the paths in T connecting the endpoints of each nontree edge.
5. Assign unit cost to each edge in the graph.
6. Repeat until terminated:
   (a) Construct ϵ-nets A'_i and B'_i for each A_i and B_i.
   (b) Let S be the set of swaps involving only ϵ-net members. Find the optimal solution $(\mathbf{p}, \delta)$ for the constraints from S together with the parameter constraints.
   (c) Find the maximum weight a_i of an edge in each A_i and the minimum

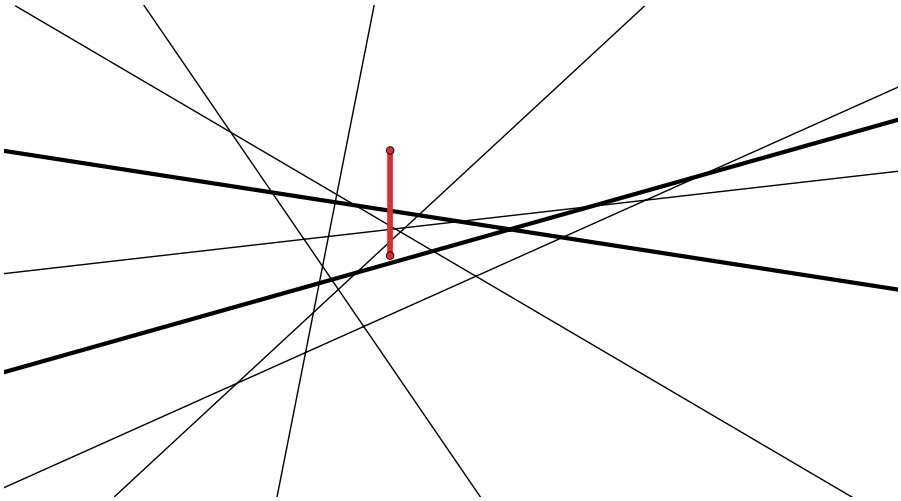


FIG. 2. *Example ϵ-net for $\epsilon = 1/2$: every vertical line segment that crosses $\geq n/2$ lines in the overall arrangement also crosses at least one of the two heavy lines.*

weight b_i of an edge in each B_i, where weights are measured according to $\mathbf{p}$. If $a_i \leq b_i - \delta$ for each i, terminate the algorithm.

(d) Find the maximum weight a'_i of an edge in each A'_i and the minimum weight b'_i of an edge in each B'_i. Double the cost of each edge in A_i with $w(e) > a'_i$, and each edge in B_i with $w(e) < b'_i$.

As in previous work on derandomized iterated reweighting schemes [7], each iteration increases the total cost of all the sets A_i (and similarly B_i) by a factor of at most $1 + \epsilon = 1 + 1/4d$, but at least one edge from the optimal base has its cost increase, so the cost of the optimal base increases on average by a factor of at least $2^{1/2d} \approx 1 + \ln(2)/2d \approx 1/2.88d$ per iteration. Since the base's cost increases at a rate faster than the total cost, it can only continue to do so for $O(d \log n)$ iterations before it overtakes the total cost—an impossibility. So at some point within those $O(d \log n)$ iterations the algorithm must terminate the loop.

THEOREM 5. *We can solve the inverse parametric minimum spanning tree problem for any constant number of parameters in worst-case time $O(m \log n \log_{m/n} n)$.*

Proof. We use the algorithm described above, setting $z = \max(2, \sqrt{m/n})$. Therefore, the total size of the sets A_i and B_i (and the total time to find these sets and to perform each iteration) is $O(m \log_{m/n} n)$. Since d is constant, there are $O(\log n)$ iterations, and the total time is $O(m \log n \log_{m/n} n)$. □

3. Other optimal subgraph problems. We now describe a method for solving inverse parametric optimization on a more general class of *optimal subgraph problems*, in which we are given a graph with parametric edge weights and must find the minimum weight *suitable subgraph*, where suitability is defined according to the particular problem. The minimum spanning tree problem considered earlier has this form, with the suitable subgraphs simply being trees. The shortest path and minimum weight matching problems also have this form. In order to solve these problems, we resort to the ellipsoid method from linear programming. This has the disadvantage of being not strongly polynomial nor very practical, but its advantages are in its extreme generality—not only can we handle any optimal subgraph problem for which the op-

timization version is polynomial, but (unlike our minimum spanning tree algorithms) we are not limited to a fixed number of parameters.

A good introduction to the ellipsoid method and its applications in combinatorial optimization can be found in the book by Grötschel, Lovász, and Schrijver [15].

LEMMA 6 (Grötschel, Lovász, and Schrijver [15, p. 158]). *For any polyhedron P defined by a strong separation oracle, and for any rational linear objective function f, one can find the point in P maximizing f in time polynomial in the dimension of P and in the maximum encoding length of the linear inequalities defining P.*

The *strong separation oracle* required by this result is a routine that takes as input a d-dimensional point and either determines that the point is in P or returns a closed halfspace containing P and not containing the test point. One slight technical difficulty with this approach is that it requires the polyhedron to be closed (else one could not separate it from a point on one of its boundary facets), while our problems are defined by strict inequalities forming open halfspaces. To solve this problem, as in the inverse spanning tree problem, we introduce an additional parameter δ measuring the separation of the desired optimal subgraph from other subgraphs and attempt to maximize δ.

THEOREM 7. *Let (G, X) be an inverse parametric optimization problem in which G is a graph with parametric edge weights and X is the given solution for an optimal subgraph problem, and in which there exists a polynomial time algorithm that determines whether X is the unique optimal subgraph and, if not, finds a different optimal subgraph Y. Then we can solve the inverse parametric optimization problem for (G, X) in time polynomial in the number of parameters, in the size of the graph, and in the maximum encoding length of the linear functions defining the edge weights of G.*

Proof. We define a polyhedron P by linear inequalities $w(X) \le w(Y) - \delta$, where w denotes the weight of a subgraph for the given point $\mathbf{p}$, Y can be any suitable subgraph, and δ is an additional parameter. To avoid problems with unboundedness, we can also introduce additional normalizing inequalities $-\mathbf{1} \le \mathbf{p} \le \mathbf{1}$. Clearly, there exists a point $(\mathbf{p}, \delta)$ with $\delta > 0$ in P if and only if $\mathbf{p}$ gives a feasible solution to the inverse parametric optimization problem.

Although there can be exponentially many inequalities, we can easily define an oracle that either terminates the entire algorithm successfully or acts as a strong separation oracle: to test a point $(\mathbf{p}, \delta)$, simply compute the optimal subgraph Y for the weights defined by $\mathbf{p}$. If $X = Y$, we have solved the problem. If $w(X) \le w(Y) - \delta$, the point is feasible. Otherwise, return the halfspace $w(X) \le w(Y) - \delta$.

Therefore, we can apply the ellipsoid method to find the point maximizing δ on P. If the method returns a point with $\delta > 0$ or terminates early with $X = Y$, we must have solved the problem; otherwise the problem must be infeasible. □

COROLLARY 8. *We can solve the inverse parametric minimum spanning tree, shortest path, or matching problems in time polynomial in the size of the given graph and in the encoding length of its parametric weight functions.*

As a variant of this result, by using an algorithm for finding the *second* best subgraph, we can test whether a given solution $(\mathbf{p}, \delta)$ is feasible even when the optimal subgraph for $\mathbf{p}$ is the desired subgraph by testing whether its weight is separated by at least δ from the next best subgraph for $\mathbf{p}$. Thus, we can complete the ellipsoid method without early termination and find a parameter value for which X is optimally separated from other subgraphs. Efficient second-best algorithms are known for minimum spanning trees [10, 17, 18], shortest paths [16], and matching [22]; in

general the second-best subgraph is the best subgraph within all graphs formed by deleting one edge of X from G.

4. Game tree search. As described in the introduction, we would like to be able to tune the weights of a game program's evaluation function so that a shallow search (to some fixed depth D) makes the correct move for each position in a given test suite. However, because of the possibility of making the right move for the wrong reasons, this problem seems highly nonlinear. Instead, we now sketch how to use the ellipsoid method to search for an evaluation function that not only makes all the right moves but also correctly orders certain pairs of positions found in its searches.

Define an *unavoidable set* of positions for a given player and depth D to be a set of positions, each of which occurs D half-moves from the present situation, such that, no matter what the opponent does, the given player can force the game to reach some position in the set. For any given position p_i, one can prove that one particular move μ is best (from the point of view of a depth D search) by exhibiting a pair (A_i, B_i) where A_i is an unavoidable set for the first player after move μ, B_i is an unavoidable set for the second player after all other moves, and the evaluation of all positions in A is better than the evaluation of any position in B. Minimax or alpha-beta search can be interpreted as finding both of these sets.

Our separation oracle runs a depth-D search on each test position until one is found at which the wrong move is made. We then construct A_i and B_i for that one position, using a deep search, and look for $a \in A_i$ and $b \in B_i$ that have evaluations in the wrong order. The oracle returns a constraint that the evaluation of a should be greater than the evaluation of b.

The result of applying the ellipsoid method with this separation oracle is either an evaluation function that makes the correct move (with a depth D search) in each test position or a determination that it is impossible to correctly order all (a, b) pairs found by the oracle.

5. Conclusions. We have discussed several problems of inverse parametric optimization, provided general solutions to a wide class of optimal subgraph problems based on the ellipsoid method, and provided faster combinatorial algorithms for the inverse parametric minimum spanning tree problem.

One difficulty with our approach comes from infeasible inputs: what if there is no linear combination of parameters that leads to the desired solution? Rogers and Langley [23] observe a similar phenomenon in their vehicle routing experiments and suggest searching for additional parameters to use. This search may be aided by the fact that infeasible linear programs can be witnessed by a small number of mutually inconsistent constraints: in the path planning problem, we can find $d+1$ paths, one of which must be better than the given path for any combination of known parameters. Studying these paths may reveal the nature of the missing parameters. Alternatively, a search for a linear programming solution with few violated constraints [19] may provide a parameter setting for which the user's chosen solution is near-optimal.

A natural direction for future research is in dealing with nonlinearity. Problems in which the solution weight includes low-degree combinations of element weights (as are used in game programming to represent interactions between positional features) may be dealt with by including additional parameters for each such combination. But what about problems in which the element weights are nonlinear combinations of the parameters? For instance, if the parameters are coordinates of points, any problem involving comparisons of distances will involve quadratic functions of those coordinates. The question of finding coordinates such that a given tree is the Euclidean

minimum spanning tree of the points is known to be NP-hard [11], but if the points' coordinates depend only on a constant number of parameters, one can solve the problem in polynomial time. Can the exponent of this polynomial be made independent of the number of parameters?

It may be possible to extend our spanning tree methods to other matroids. For example, transversal matroids provide a formulation of bipartite matching in which the weights are on the vertices of one side of the bipartition rather than the edges. Can we solve inverse parametric transversal matroid optimization efficiently? Are there natural applications of this or other matroidal problems?

Another open question concerns the existence of combinatorial algorithms for the inverse parametric shortest path problem. It is unlikely that a strongly polynomial algorithm exists without restricting the dimension: one can encode any linear programming feasibility problem as an inverse parametric shortest path (or other optimal subgraph) problem by using a parallel pair of edges for each constraint. But is there a strongly polynomial algorithm for inverse parametric shortest paths when the number of parameters is small?

REFERENCES

[1] P. K. Agarwal, D. Eppstein, L. J. Guibas, and M. R. Henzinger, *Parametric and kinetic minimum spanning trees*, in Proceedings of the 39th Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 596–605.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice–Hall, Englewood Cliffs, NJ, 1993.

[3] R. K. Ahuja and J. B. Orlin, *Combinatorial Algorithms for Inverse Network Flow Problems*, Working paper SWP 4004, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, 1998.

[4] R. K. Ahuja and J. B. Orlin, *Inverse Optimization* I: *Linear Programming and General Problem*, Working paper SWP 4002, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, 1998.

[5] R. K. Ahuja and J. B. Orlin, *Inverse Optimization* II: *Network Flow Problems*, Working paper SWP 4003, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, 1998.

[6] H. Brönnimann, B. Chazelle, and J. Matoušek, *Product range spaces, sensitive sampling, and derandomization*, SIAM J. Comput., 28 (1999), pp. 1552–1575.

[7] H. Brönnimann and M. T. Goodrich, *Almost optimal set covers in finite VC-dimension*, Discrete Comput. Geom., 14 (1995), pp. 463–479.

[8] T. M. Chan, *Geometric applications of a randomized optimization technique*, Discrete Comput. Geom., 22 (1999), pp. 547–567.

[9] K. L. Clarkson, *Las Vegas algorithms for linear and integer programming when the dimension is small*, J. ACM, 42 (1995), pp. 488–499.

[10] B. Dixon, M. Rauch, and R. E. Tarjan, *Verification and sensitivity analysis of minimum spanning trees in linear time*, SIAM J. Comput., 21 (1992), pp. 1184–1192.

[11] P. Eades and S. Whitesides, *The realization problem for Euclidean minimum spanning trees is NP-hard*, Algorithmica, 16 (1996), pp. 60–82.

[12] G. N. Frederickson, *Data structures for on-line updating of minimum spanning trees with applications*, SIAM J. Comput., 14 (1985), pp. 781–798.

[13] G. N. Frederickson, *Ambivalent data structures for dynamic* 2*-edge-connectivity and* k *smallest spanning trees*, SIAM J. Comput., 26 (1997), pp. 484–538.

[14] J. A. Garay, S. Kutten, and D. Peleg, *A sublinear time distributed algorithm for minimum-weight spanning trees*, SIAM J. Comput., 27 (1998), pp. 302–316.

[15] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Algorithms Combin. 2, Springer-Verlag, Berlin, 1988.

[16] W. Hoffman and R. Pavley, *A method for the solution of the* N*th best path problem*, J. ACM, 6 (1959), pp. 506–514.

[17] N. Katoh, T. Ibaraki, and H. Mine, *An algorithm for finding* K *minimum spanning trees*, SIAM J. Comput., 10 (1981), pp. 247–255.

[18] V. King, *A simpler minimum spanning tree verification algorithm*, Algorithmica, 18 (1997), pp. 263–270.
[19] J. Matoušek, *On geometric optimization with few violated constraints*, Discrete Comput. Geom., 14 (1995), pp. 365–384.
[20] N. Megiddo, *Applying parallel computation algorithms in the design of sequential algorithms*, J. ACM, 30 (1983), pp. 852–865.
[21] N. Megiddo, *Linear programming in linear time when the dimension is fixed*, J. ACM, 31 (1984), pp. 114–127.
[22] L. Mihályffy, *On the problem of the second best assignment*, Problems Control Inform. Theory, 8 (1979), pp. 257–265.
[23] S. Rogers and P. Langley, *Interactive refinement of route preferences for driving*, in Proceedings of the Spring Symposium on Interactive and Mixed-Initiative Decision-Theoretic Systems, AAAI Press, Menlo Park, CA, 1998, pp. 109–113.
[24] P. T. Sokkalingam, R. K. Ahuja, and J. B. Orlin, *Solving inverse spanning tree problems through network flow techniques*, Oper. Res., 47 (1999), pp. 291–298.
[25] J. Stanback, *Chess GA experiments*. Message to rec.games.chess.computer, 1996.
[26] N. E. Young, R. E. Tarjan, and J. B. Orlin, *Faster parametric shortest path and minimum-balance algorithms*, Networks, 21 (1991), pp. 205–221.
[27] J. Zhang, Z. Liu, and Z. Ma, *On the inverse problem of minimum spanning tree with partition constraints*, Math. Methods Oper. Res., 44 (1996), pp. 171–187.
[28] J. Zhang and Z. Ma, *A network flow method for solving some inverse combinatorial optimization problems*, Optimization, 37 (1996), pp. 59–72.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 654–670

AN APPROXIMATION ALGORITHM FOR MINIMUM CONVEX COVER WITH LOGARITHMIC PERFORMANCE GUARANTEE*

STEPHAN J. EIDENBENZ† AND PETER WIDMAYER‡

Abstract. The problem MINIMUM CONVEX COVER of covering a given polygon with a minimum number of (possibly overlapping) convex polygons is known to be *NP*-hard, even for polygons without holes [J. C. Culberson and R. A. Reckhow, *J. Algorithms,* 17 (1994), pp. 2–44]. We propose a polynomial-time approximation algorithm for this problem for polygons with or without holes that achieves an approximation ratio of $O(\log n)$, where n is the number of vertices in the input polygon. To obtain this result, we first show that an optimum solution of a restricted version of this problem, where the vertices of the convex polygons may lie only on a certain grid, contains at most three times as many convex polygons as the optimum solution of the unrestricted problem. As a second step, we use dynamic programming to obtain a convex polygon which is maximum with respect to the number of "basic triangles" that are not yet covered by another convex polygon. We obtain a solution that is at most a logarithmic factor off the optimum by iteratively applying our dynamic programming algorithm. Furthermore, we show that MINIMUM CONVEX COVER is *APX*-hard; i.e., there exists a constant $\delta > 0$ such that no polynomial-time algorithm can achieve an approximation ratio of $1+\delta$. We obtain this result by analyzing and slightly modifying an already existing reduction [J. C. Culberson and R. A. Reckhow, *J. Algorithms,* 17 (1994), pp. 2–44].

Key words. convex cover, art gallery, inapproximability, approximation algorithms, dynamic programming

AMS subject classifications. 68U05, 68W25, 68Q25

PII. S0097539702405139

1. Introduction and problem definition.

The problem MINIMUM CONVEX COVER is the problem of covering a given polygon T with a minimum number of (possibly overlapping) convex polygons that lie in T. This problem belongs to the family of classic art gallery problems; it is known to be *NP*-hard for input polygons with holes [17] and without holes [4]. The study of approximations for hard art gallery problems has rarely led to good algorithms or good lower bounds; we discuss a few exceptions below. In this paper, we propose the first nontrivial approximation algorithm for MINIMUM CONVEX COVER. Our algorithm works for polygons with and without holes. It relies on a strong relationship between the continuous original problem version and a particular discrete version in which all relevant points are restricted to lie on a kind of grid that we call a quasi grid. The quasi grid is the set of intersection points of

*Received by the editors April 8, 2002; accepted for publication (in revised form) December 27, 2002; published electronically April 17, 2003. Preliminary versions of this paper have appeared in conference proceedings [in Algorithms—ESA 2001 (Århus), Lecture Notes in Comput. Sci. 2161, Springer-Verlag, Berlin, 2001, pp. 333–343] and in Stephan Eidenbenz's thesis [Ph.D. thesis, Dissertation ETH 13683, Zürich, Switzerland, 2000]. This work was partially supported by the Swiss Federal Office for Education and Science under contract BBW 01.0068 within a project funded by the European Union under contract IST-2001-32007 (APPOL II). This work was performed by an employee of the U.S. Government or under U.S. Government contract. The U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

http://www.siam.org/journals/sicomp/32-3/40513.html

†P.O. Box 1663, MS M997, Los Alamos National Laboratory, Los Alamos, NM 87545 (eidenben@lanl.gov) LA-UR:02-7641. This author's work was done while he was at the Institute of Theoretical Computer Science, Swiss Federal Institute of Technology, Zürich, Switzerland.

‡Institute of Theoretical Computer Science, Swiss Federal Institute of Technology, 8092 Zürich, Switzerland (widmayer@inf.ethz.ch).

all lines connecting two vertices of the input polygon. In the RESTRICTED MINIMUM CONVEX COVER problem, the vertices of the convex polygons that cover the input polygon may lie only on this quasi grid. We prove that an optimum solution of the RESTRICTED MINIMUM CONVEX COVER problem needs at most three times the number of convex polygons that the MINIMUM CONVEX COVER solution needs. To find an approximate solution for the RESTRICTED MINIMUM CONVEX COVER problem, we propose a greedy approach: we compute one convex polygon of the solution after the other, and we pick as the next convex polygon one that covers a maximum number of triangles defined on an even finer quasi grid, where these triangles are not yet covered by previously chosen convex polygons. We propose an algorithm for finding such a maximum convex polygon by means of dynamic programming. To obtain an upper bound on the quality of the solution, we interpret our covering problem on triangles as a special case of the general MINIMUM SET COVER problem that gives as input a base set of elements and a collection of subsets of the base set and that asks for a smallest number of subsets in the collection whose union contains all elements of the base set. In our special case, each triangle is an element, and each possible convex polygon is a possible subset in the collection, but not all of these subsets are represented explicitly. (There could be an exponential number of subsets.) This construction translates the logarithmic quality of the approximation from MINIMUM SET COVER to MINIMUM CONVEX COVER [13].

On the negative side, we show that MINIMUM CONVEX COVER is *APX*-hard; i.e., there exists a constant $\delta > 0$ such that no polynomial-time algorithm can achieve an approximation ratio of $1 + \delta$ (see [3] for an introduction to the class *APX*). This inapproximability result is based on a problem transformation shown by Culberson and Reckhow [4]; we modify this transformation and show that it is gap-preserving (as defined by Arora and Lund [1]).

The related problem of partitioning a given polygon into a minimum number of nonoverlapping convex polygons is polynomially solvable for input polygons without holes [2]. It is *NP*-hard for input polygons with holes [15] and can be approximated with an approximation ratio of 4 [12]; it remains *NP*-hard even if the convex partition must be created by cuts from a given family of (at least three) directions [16]. Other related results for art gallery problems include approximation algorithms with logarithmic approximation ratios for MINIMUM VERTEX GUARD and MINIMUM EDGE GUARD [10], as well as for the problem of covering a polygon with rectangles in any orientation [11]. Furthermore, logarithmic inapproximability results are known for MINIMUM POINT/VERTEX/EDGE GUARD for polygons with holes, and *APX*-hardness results are known for the same problems for polygons without holes [6]. The related problem RECTANGLE COVER of covering a given orthogonal polygon with a minimum number of rectangles can be approximated with a constant ratio for polygons without holes [9] and with an approximation ratio of $O(\sqrt{\log n})$ for polygons with holes [14]. For additional results, see the surveys on art galleries [18, 19]. The general idea of using dynamic programming to find maximum convex structures has been used before to solve the problem of finding a maximum (with respect to the number of vertices) empty convex polygon, given a set of vertices in the plane [5]. An $O(\log n)$ approximation algorithm for the problem of covering a polygon with rectangles in any orientation [11] relies on an approach similar to ours that consists of making the problem discrete and then transforming it to a MINIMUM SET COVER instance.

In section 2, we define the quasi grid and its refinement into triangles. Section 3

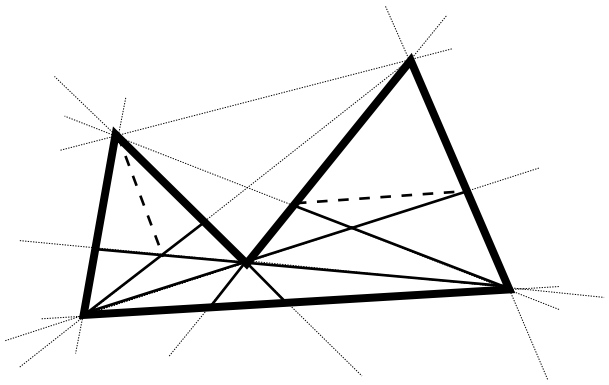


FIG. 1. *Construction of first-order basic triangles.*

contains the proof of the linear relationship between the sizes of the optimum solutions of the unrestricted and restricted convex cover problems. We propose a dynamic programming algorithm to find a maximum convex polygon in section 4 before showing how to iteratively apply this algorithm to find a convex cover in section 5. In section 6, we present the proof of the *APX*-hardness of MINIMUM CONVEX COVER. Concluding thoughts are in section 7.

2. From the continuous to the discrete. We consider simple input polygons with and without holes, where a polygon T is given as an ordered list of vertices in the plane. If T contains holes, each hole is also given as an ordered list of vertices. Let V_T denote the set of vertices (including the vertices of holes, if any) of a given polygon T. While in the general MINIMUM CONVEX COVER problem the vertices of the convex polygons that cover the input polygon can be positioned anywhere in the interior or on the boundary of the input polygon, we restrict their positions in an intermediate step: they may be positioned only on a quasi grid in the RESTRICTED MINIMUM CONVEX COVER problem.

In order to define the RESTRICTED MINIMUM CONVEX COVER problem more precisely, we partition the interior of a polygon T into *convex components* (as proposed in [10] for a different purpose) by drawing a line through each pair of vertices of T. We then triangulate each convex component arbitrarily. We call the triangles thus obtained *first-order basic triangles*. Figure 1 shows an example of the first-order basic triangles of a polygon (thick solid lines) with an arbitrary triangulation (fine solid lines and dashed lines). If a polygon T consists of n vertices, drawing a line through each pair of vertices of T will yield less than $\binom{n}{2} \cdot \binom{n}{2} \in O(n^4)$ intersection points. Let V_T^1 be the set of these intersection points that lie in T (in the interior or on the boundary). Note that $V_T \subseteq V_T^1$. The first-order basic triangles are a triangulation of V_T^1 inside T; therefore, the number of first-order basic triangles is also $O(n^4)$. The RESTRICTED MINIMUM CONVEX COVER problem asks for a minimum number of convex polygons, with vertices restricted to V_T^1, that together cover the input polygon T. We call V_T^1 a quasi grid that is imposed on T. For solving the RESTRICTED MINIMUM CONVEX COVER problem, we make use of a finer quasi grid: simply partition T by drawing lines through each pair of points from V_T^1. This yields again convex components, and we triangulate them again arbitrarily. This higher resolution partition yields $O(n^{16})$ intersection points, which define the set V_T^2. We call the resulting triangles *second-order basic triangles*. Obviously, there are $O(n^{16})$ second-order basic triangles. Note

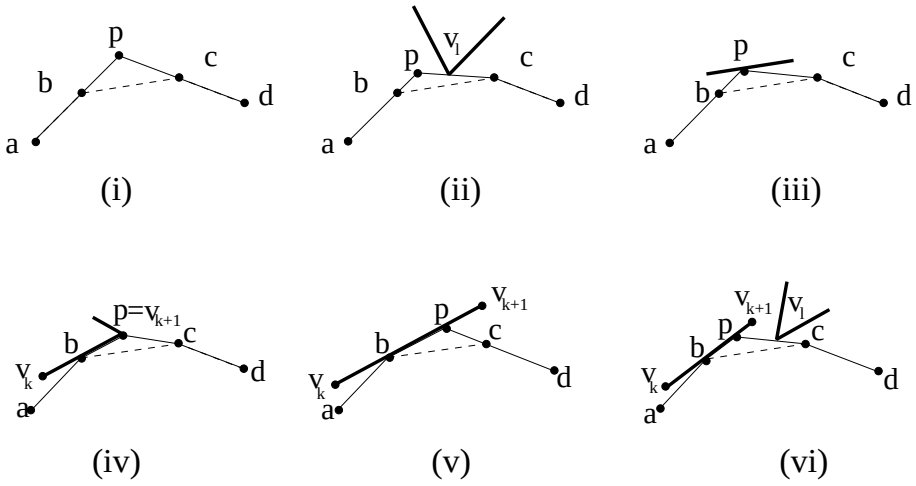


FIG. 2. *Expansion of edge (b,c).*

that $V_T \subseteq V_T^1 \subseteq V_T^2$.

3. The optimum solution of MINIMUM CONVEX COVER vs. the optimum solution of RESTRICTED MINIMUM CONVEX COVER. The quasi grids V_T^1 and V_T^2 serve the purpose of making a convex cover computationally efficient while at the same time guaranteeing that the cover on the discrete quasi grid is not much worse than the desired cover in continuous space. The following theorem proves the latter.

THEOREM 1. *Let T be an arbitrary simple input polygon with n vertices. Let OPT denote the size of an optimum solution of* MINIMUM CONVEX COVER *with input polygon T, and let OPT' denote the size of an optimum solution of* RESTRICTED MINIMUM CONVEX COVER *with input polygon T. Then*

$$OPT' \leq 3 \cdot OPT.$$

Proof. We proceed as follows: we show how to *expand* a given arbitrary convex polygon $C \subseteq T$ to another convex polygon $C' \subseteq T$ with $C \subseteq C'$ by iteratively expanding edges. We then replace the vertices in C' by vertices from V_T^1, which results in a (possibly) nonconvex polygon $C'' \subseteq T$ with $C' \subseteq C''$. Finally, we describe how to obtain three convex polygons C_1'', C_2'', C_3'' with $C'' = C_1'' \cup C_2'' \cup C_3''$ that contain only vertices from V_T^1. This will complete the proof, since each convex polygon from an optimum solution of MINIMUM CONVEX COVER can be replaced by at most three convex polygons that are in a solution of RESTRICTED MINIMUM CONVEX COVER. Following this outline, let us present the proof details.

Expanding edges. Let C be an arbitrary convex polygon inside polygon T. Let the vertices of C be given in clockwise order. We obtain a series of convex polygons $C^1, C^2, \ldots, C'$ with $C = C^0 \subseteq C^1 \subseteq C^2 \subseteq \cdots \subseteq C'$, where C^{i+1} is obtained from C^i as follows (see Figure 2).

Let a, b, c, d be consecutive vertices (in clockwise order) in the convex polygon C^i that lies inside polygon T. For ease of description, we assume that C^i does not contain vertices that are collinear with its two neighboring vertices, except when such a vertex happens to be a vertex from V_T; moreover, any vertex from V_T that lies on the boundary of C^i is also a vertex of C^i, even if it has collinear neighbors in C^i.

Let vertices $b, c \notin V_T$, with b and c not on the same edge of T. Then the edge (b, c) is called *expandable*. If there exists no expandable edge in C^i, then $C' = C^i$, which means that we have found the end of the series of convex polygons. If (b, c) is an expandable edge, we *expand* the edge from vertex b to vertex c as follows:

- If b does not lie on the boundary of T, then we let a point p start at b and move along the halfline through a and b away from a and b until either one of the following two events happens: p lies on the line through c and d, or the triangle p, c, b touches the boundary of T. Fix p as soon as the first of these events happens. Figure 2 shows a list of all possible cases, where the edges from polygon T are drawn as thick edges: point p lies on the intersection point of the lines from a through b and from c through d as in case (i), or there is a vertex v_l on the line segment from p to c as in case (ii), or p lies on an edge of T as in case (iii).
- If b lies on the boundary of T, i.e., on some edge of T, say, from v_k to v_{k+1} (in clockwise order), then let p move from b as before, except that the move is now along the halfline from v_k through b away from v_k and b up until at most v_{k+1} (instead of the ray from a through b). Figure 2 shows a list of all possible cases: point p lies either at vertex v_{k+1} as in case (iv) or on the intersection point of the lines from b to v_{k+1} and from d through c as in case (v), or there is a vertex v_l on the line segment from p to c as in case (vi).

A new convex polygon C_p^i is obtained by simply adding point p as a vertex in the ordered set of vertices of C^i between the two vertices b and c; if—as in cases (ii) and (vi)—a vertex from V_T lies on the boundary of C_p^i, it is also added as a vertex (despite the fact that it may have two collinear neighbors). In contrast, all vertices in C_p^i that have collinear neighbors and that are not vertices in V_T are eliminated.

An edge from two consecutive vertices b and c with $b, c \notin V_T$ can always be expanded in such a way that the triangle b, p, c that is added to the convex polygon is nondegenerate, i.e., has nonzero area, unless b and c both lie on the same edge of polygon T. This follows from the cases (i)–(vi) of Figure 2.

Let $C^{i+1} = C_p^i$ if either a new vertex of V_T has been added to C_p^i in the expansion of the edge, which is true in cases (ii), (iv), and (vi), or the number of vertices of C_p^i that are not vertices from V_T has decreased, which is true in case (i). If p is as in case (iii), we expand the edge (p, c), which will result in case (iv), (v), or (vi). Note that in cases (iv) and (vi), we have found C^{i+1}. If p is as in case (v), we expand the edge (p, d), which will result in case (iv), (v), or (vi). If it is case (v) again, we repeat the procedure by expanding the edge from p and the successor (clockwise) of d. This needs to be done at most as many times as there are vertices in C^i, since the procedure eliminates a vertex from C^i in each iteration and will stop before it tries to expand an edge ending at vertex a as the resulting polygon would not be convex. Therefore, we obtain C^{i+1} from C^i in a finite number of steps.

Let τ_i denote the number of vertices in C^i that are also vertices in T, and let $\hat{\tau}_i$ be the number of vertices in C^i that are not vertices in T. Note that $\phi(i) = \hat{\tau}_i - 2\tau_i + 2n$ is a function that bounds the number of remaining iteration steps that are needed to reach C'; it strictly decreases with every increase in i and cannot become negative as $\hat{\tau}_i$ and τ_i are both nonnegative numbers by definition and $n \geq \tau_i$. The existence of such a bounding function, which is often called a variant function, implies the finiteness of the series $C^1, C^2, \ldots, C'$ of convex polygons.

By definition, there are no expandable edges left in C'. Call a vertex of C' a T-vertex if it is a vertex in T. From the definition of expandable edges, it is clear that

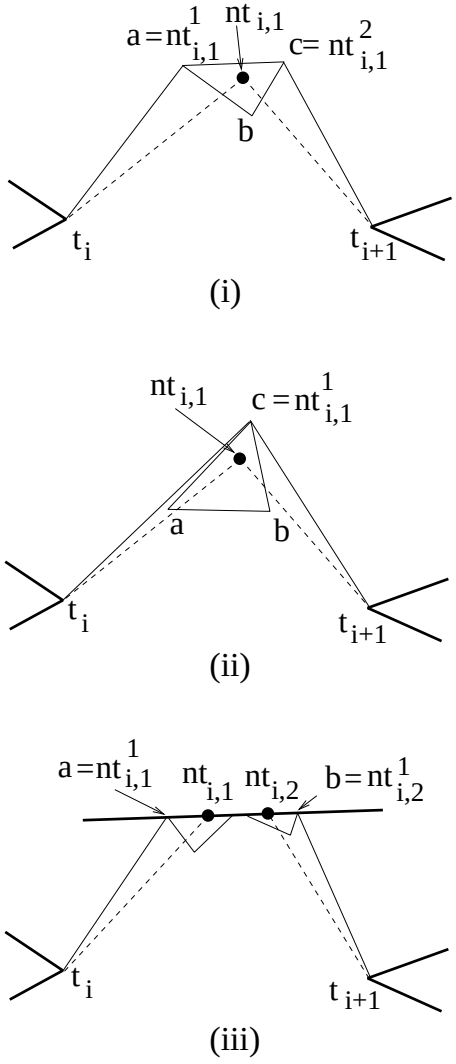


FIG. 3. *Replacing non-T-vertices.*

there can be at most two non-T-vertices between any two consecutive T-vertices in C', and if there are two non-T-vertices between two consecutive T-vertices, they must both lie on the same edge in T. (Otherwise, the edge between two non-T-vertices would be expandable, which contradicts the definition of C'.)

Replacing vertices. Let the T-vertices in C' be $t_1, \ldots, t_l$ in clockwise order, and let the non-T-vertices between t_i and t_{i+1} be $nt_{i,1}$ and $nt_{i,2}$ if they exist. We will replace each non-T-vertex $nt_{i,j}$ in C' by one or two vertices $nt^1_{i,j}$ and $nt^2_{i,j}$ that are both elements of the quasi grid V^1_T. This will transform the convex polygon C' into a not necessarily convex polygon C''. (We will show later how C'' can be covered by at most three convex polygons C''_1, C''_2, C''_3.) The details are as follows: let a, b, c be the first-order basic triangle in which non-T-vertex $nt_{i,j}$ lies, as illustrated in Figure 3. Points a, b, c are all visible from both vertices t_i and t_{i+1}. To see this, assume by contradiction that the view from, say, t_i to a is blocked by an edge e of T. Since $nt_{i,j}$

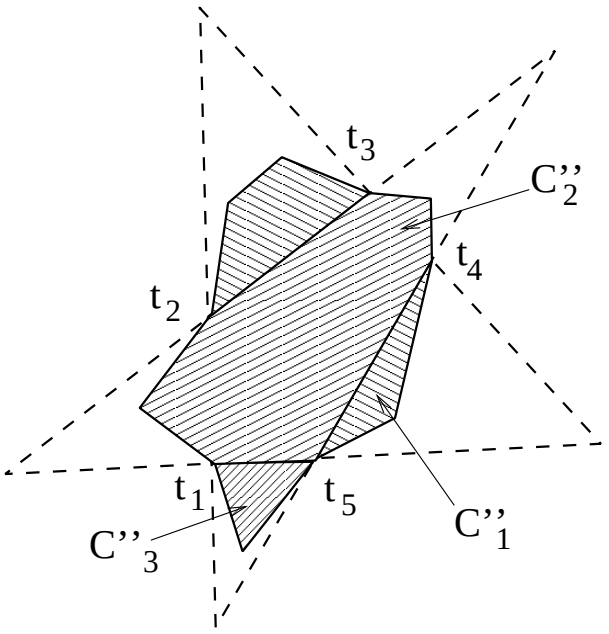


FIG. 4. *Covering C'' with three convex polygons.*

must see t_i, the edge e must contain a vertex e' in the triangle $t_i, a, nt_{i,j}$, but then a cannot be a vertex of the first-order basic triangle in which $nt_{i,j}$ lies, since the line from vertex t_i through vertex e' would cut through the first-order basic triangle—an impossibility.

Assume that only one non-T-vertex $nt_{i,1}$ exists between t_i and t_{i+1}. If the triangle t_i, t_{i+1}, a completely contains the triangle $t_i, nt_{i,1}, t_{i+1}$, then we let $nt^1_{i,1} = a$, and likewise for b and c (see Figure 3 (ii)). Otherwise, we let $(nt^1_{i,1}, nt^2_{i,1})$ be $(a, b), (a, c)$, or (b, c), as in Figure 3 (i), such that the polygon $t_i, nt^1_{i,1}, nt^2_{i,1}, t_{i+1}$ is convex and completely contains the triangle $t_i, nt_{i,1}, t_{i+1}$. This is always possible by the definition of points a, b, c.

Assume that two non-T-vertices $nt_{i,1}$ and $nt_{i,2}$ exist between t_i and t_{i+1}. From the definition of C', we know that $nt_{i,1}$ and $nt_{i,2}$ must lie on the same edge e of T. Therefore, the basic triangle in which $nt_{i,1}$ lies must contain a vertex a either at $nt_{i,1}$ or preceding $nt_{i,1}$ on edge e along T in clockwise order. Let $nt^1_{i,1} = a$. The basic triangle in which $nt_{i,2}$ lies must contain a vertex b either at $nt_{i,2}$ or succeeding $nt_{i,2}$ on edge e. Let $nt^1_{i,2} = b$. (See Figure 3 (iii).) Note that the convex polygon $t_i, nt^1_{i,1}, nt^1_{i,2}, t_{i+1}$ completely contains the polygon $t_i, nt_{i,1}, nt_{i,2}, t_{i+1}$.

After applying this change to all non-T-vertices in C', we obtain a (possibly) nonconvex polygon C''.

Covering with three convex polygons. We will now show how to cover C'' with at most three convex polygons. First, assume that C'' contains an odd number f of T-vertices. We let C''_1 be the polygon defined by vertices $t_i, nt^k_{i,j}$, and t_{i+1} for all j, k and for all odd i, but $i \neq f$. By construction, C''_1 is convex. To see this, assume C''_1 is not convex; it would then have to have at least one vertex whose inner angle is larger than π, which cannot happen at non-T-vertices in C''_1 by construction. The inner angles at a T-vertex t_{i-1} for i odd cannot be larger than π either, because

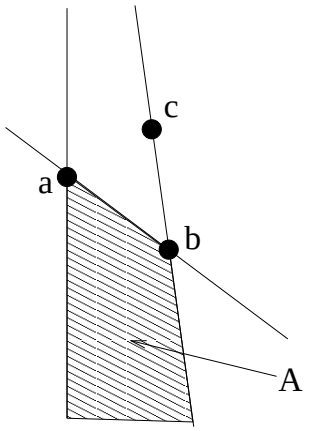


FIG. 5. *Dynamic programming.*

polygon C_1'' lies entirely to the right of the line going from t_{i-1} through t_i. Let C_2'' be the polygon defined by vertices $t_i, nt_{i,j}^k$, and t_{i+1} for all j, k and for all even i. Finally, let C_3'' be the polygon defined by vertices $t_f, nt_{f,j}^k$, and t_1 for all j, k. Using similar arguments as for C_1'', polygons C_2'' and C_3'' are convex as well. Figure 4 shows an example. Obviously, C_1'', C_2'', and C_3'' together cover all of C''. Second, assume that C'' contains an even number of T-vertices, and cover it with only two convex polygons using the same concept. This completes the proof. □

4. Finding maximum convex polygons. Assume that each second-order basic triangle from a polygon T is assigned a weight value of either 1 or 0. In this section, we present an algorithm using dynamic programming that computes a convex polygon M in a polygon T that contains a maximum number of second-order basic triangles with weight 1 and that has vertices only from V_T^1. For simplicity, we call such a polygon a *maximum convex polygon*. The weight of a polygon M is defined as the sum of the weights of the second-order basic triangles in the polygon and is denoted by $|M|$. We will later use the algorithm described below to iteratively compute a maximum convex polygon with respect to the triangles that are not yet covered, to eventually obtain a convex cover for T.

Let $a, b, c \in V_T^1$. Let $P_{a,b,c}$ denote the maximum convex polygon that

- contains only vertices from V_T^1,
- contains vertices a, b, c in counterclockwise order,
- has a as its left-most vertex,[1]
- contains additional vertices only between vertices a and b, and
- is completely contained in T.

Given three vertices $a, b, c \in V_T^1$, let A be the (possibly infinite) area of points that are

- to the right of vertex a,
- to the left of the line oriented from b through a, and
- to the left of the line oriented from b through c.

[1] If polygon $P_{a,b,c}$ has several left-most vertices, vertex a is one of them.

1.	Initialize table $S(a,b,c)$ with zeros	
2.	FORALL $a \in V_T^1$ DO	$O(n^{28})$
3.	Choose a helper point a' with the same x-coordinate and an arbitrary but smaller y-coordinate than a	
4.	Order all vertices $b \in V_T^1$ to the right of a according to the angle formed by b, a, a'; let the resulting ordered set be B	$O(n^4 \log n)$
5.	$B' := \emptyset$	
6.	WHILE $B \neq \emptyset$ DO	$O(n^{24})$
7.	Let b be the smallest element in B; $B := B - \{b\}$; $B' := B' \cup \{b\}$	
8.	FORALL $c \in V_T^1 \backslash B'$ to the right of a DO	$O(n^{20})$
9.	Compute $\lvert\Delta a, b, c\rvert$	$O(n^{16})$
10.	Define area A with respect to vertices a, b, c according to Lemma 2	
11.	FORALL $d \in (V_T^1 \cap A)$ DO	$O(n^4)$
12.	Look up $\lvert P_{a,d,b}\rvert$ and store maximizing d in $d_{\max}$	
13.	END	
14.	$\lvert P_{a,b,c}\rvert := \lvert\Delta a, b, c\rvert + \lvert P_{a,d_{\max},b}\rvert$	
15.	Store $\lvert P_{a,b,c}\rvert$ in table S	
16.	END	
17.	END	
18.	END	
19.	Find maximum entry in table S	

FIG. 6. *Algorithm for computing a maximum weight convex polygon.*

For an illustration, see Figure 5. Let

$$P'_{a,b,c} = \max_{d \in V_T^1 \cap A} P_{a,d,b} \cup \Delta a, b, c,$$

where $\Delta a, b, c$ is the triangle a, b, c and max is defined as follows (to simplify notation):

$$\max\{P_1, P_2\} = \begin{cases} P_1 & \text{if } |P_1| \geq |P_2|, \\ P_2 & \text{otherwise.} \end{cases}$$

LEMMA 2. *$P_{a,b,c} = P'_{a,b,c}$ if the triangle a, b, c is completely contained in the polygon T.*

Proof. Consider $P_{a,b,c}$, which is maximum by definition. $P_{a,b,c}$ must contain additional vertices between a and b. (Otherwise, the lemma is trivially true.) Let d' be the predecessor of b in the counterclockwise order of $P_{a,b,c}$. Vertex d' must lie in A as defined above. Now consider $P'' = P_{a,b,c} - \Delta a, b, c$. From the definition of A it is clear that P'' can contain only vertices that lie in A. Now $P_{a,d',b}$ is maximum by definition, and it is considered when computing $P'_{a,b,c}$. □

Let M be a maximum convex polygon for a polygon T with weights assigned to the second-order basic triangles. Let a be the left-most vertex of M, let c be the predecessor of a in M in counterclockwise order, and let b be the predecessor of c. Then $|P_{a,b,c}| = |M|$ by definition. We will use Lemma 2 to construct an algorithm, which takes as input a polygon T and an assignment of weight 0 or 1 to each second-order basic triangle of T and computes the maximum convex polygon. An overview of the algorithm is given in Figure 6.

In more detail, we start by initializing a table $S(a, b, c)$, where the entry at position a, b, c denotes the weight $|P_{a,b,c}|$, in line 1 of Figure 6. In a first loop, we fix vertex $a \in V_T^1$ in line 2, let a' be a helper point with the same x-coordinate and an arbitrary but smaller y-coordinate than a, and order all vertices $b \in V_T^1$ to the right of a according to the angle formed by b, a, a'. We call the resulting ordered set B and let B' be the empty set. In a second loop, starting at line 6, we iteratively take the smallest element b from B, remove it from B, and add it to set B'; then for every $c \in V_T^1 \backslash B'$ to the right of a (see line 8), we compute weight $|\Delta a, b, c|$ of the triangle a, b, c and compute $P_{a,b,c}$ according to Lemma 2 in line 11 (i.e., look up the values of $P_{a,d,b}$ for all $d \in V_T^1 \cap A$ and take the maximum; all these values were computed in earlier iterations). We then compute weight $|P_{a,b,c}|$ by adding $|\Delta a, b, c|$ to $|P_{a,d,b}|$, where d is the maximizing argument, and store the value in table S. Note that the computation of $P_{a,b,c}$ according to Lemma 2 is always possible, since all possible vertices d in $P_{a,d,b}$ lie to the left of the line from b to a (see also definition of area A), have therefore smaller angles d, a, a' than b, a, a', and have therefore already been computed. The algorithm is executed for every $a \in V_T^1$, and—by using standard bookkeeping techniques (not explicitly given in the pseudocode of Figure 6)—the maximum convex polygon found is returned.

The cumulative running times of the loops and the running times of some crucial individual lines of the algorithm are given in Figure 6, resulting in an overall running time of $O(n^{28})$. To see this, we first look at the loop from line 8 to line 16: each iteration of this loop takes time $O(n^{16})$, which is the running time of computing the weight of a triangle a, b, c (see line 9) as we have to add the weights of almost all second-order basic triangles; the $O(n^4)$ running time of the inner loop (lines 11 to 13) is dominated by the $O(n^{16})$ running time of line 9. Since there are $O(n^4)$ iterations of the loop from line 8 to line 16, we get a running time of $O(n^{20})$ for this loop. The loop from lines 6 to 17 consists of a total of $O(n^4)$ iterations of the $O(n^{20})$ loop from lines 8 to 16, thus resulting in a cumulative running time of $O(n^{24})$. Finally, the loop from lines 2 to 18 has $O(n^4)$ iterations of the $O(n^{24})$ loop from lines 6 to 17; the $O(n^4 \log n)$ time required for sorting in line 4 is dominated by the $O(n^{24})$ time for the loop. Thus the overall running time is $O(n^{28})$. Memory requirements are $O(n^{12})$ as we need to allocate table S.

5. An approximation algorithm for Minimum Convex Cover. Given a polygon T, we obtain a convex cover by iteratively applying the algorithm for computing a maximum convex polygon from section 4. It works as follows for an input polygon T:

1. Let all second-order basic triangles have weight 1. Let $S = \emptyset$.
2. Find the maximum convex polygon M of polygon T using the algorithm from section 4, and add M to the solution S. Decrease the weight of all second-order basic triangles that are contained in M to 0.[2]
3. Repeat step 2 until there are no second-order basic triangles with weight 1 left. Return S.

To obtain a performance guarantee for this algorithm, consider the Minimum Set Cover instance I, which has all second-order basic triangles as elements and where the second-order basic triangles with weight 1 of each convex polygon in T, which contains only vertices from V_T^1, form a set in I. The greedy heuristic for Minimum

[2]Note that by the definition of second-order basic triangles, a second-order basic triangle either is completely contained in M or is completely outside M.

SET COVER achieves an approximation ratio of $1 + \ln n'$, where n' is the number of elements in I [13], and it works in exactly the same way as our algorithm. However, we do not have to (and could not afford to) compute all the sets of the MINIMUM SET COVER instance I (which would be a number exponential in n'); it suffices to always compute a set, which contains a maximum number of elements not yet covered by the solution thus far. This is achieved by reducing the weights of the second-order basic triangles already in the solution to 0; i.e., a convex polygon with maximum weight is such a set.

Note that $n' = O(n^{16})$ since the number of triangles in a triangulation is proportional to the number of points in V_T^2 that induce the triangulation. Therefore, our algorithm achieves an approximation ratio of $O(\log n)$ for RESTRICTED MINIMUM CONVEX COVER on input polygon T. Because of Theorem 1, we know that the solution found for RESTRICTED MINIMUM CONVEX COVER is also a solution for the unrestricted MINIMUM CONVEX COVER that is at most a factor of $O(\log n)$ off the optimum solution.

As for the running time of this algorithm, observe that the algorithm adds to the solution in each round a convex polygon with nonzero weight. An optimum solution would consist of at most $O(n)$ convex polygons, since a triangulation of the vertices of the input polygon yields a trivial solution with $O(n)$ convex polygons that are triangles in this case. Since our algorithm finds a solution that is at most a factor $O(\log n)$ off the optimum solution and since it adds a convex polygon to the solution in each round, there can be at most $O(n \log n)$ rounds before the algorithm finishes. As each round takes time $O(n^{28})$, the total running time is $O(n^{29} \log n)$. This completes the proof of our first main theorem:

THEOREM 3. MINIMUM CONVEX COVER *for input polygons with or without holes can be approximated by a polynomial-time algorithm with an approximation ratio of* $O(\log n)$*, where* n *is the number of polygon vertices.*

6. *APX*-hardness of MINIMUM CONVEX COVER. The upper bound of $O(\log n)$ on the approximation ratio for MINIMUM CONVEX COVER may not be tight: we will now prove that there is a constant lower bound on the approximation ratio, and hence a gap remains. More precisely, we prove MINIMUM CONVEX COVER to be *APX*-hard. Our proof of the *APX*-hardness of MINIMUM CONVEX COVER for input polygons with or without holes uses a construction similar to the one that is used to prove the *NP*-hardness of this problem for input polygons without holes[4].[3] However, we reduce the problem MAXIMUM 5-OCCURRENCE-3-SAT rather than SATISFIABILITY (SAT) (as done in the original reduction [4]) to MINIMUM CONVEX COVER, and we design the reduction to be gap-preserving [1]. MAXIMUM 5-OCCURRENCE-3-SAT is the variant of SAT in which each variable may appear at most five times in clauses and each clause contains at most three literals. MAXIMUM 5-OCCURRENCE-3-SAT is *APX*-complete [1].

The reduction is constructed as follows: for a given instance I of MAXIMUM 5-OCCURRENCE-3-SAT with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$, we construct an instance I' of MINIMUM CONVEX COVER. To stick to the notation of [4], let $l_i \leq 5$ denote the number of literals of variable x_i in the clauses, and let $l = \sum_{i=1}^{n} l_i$ be the total number of literals.

For each literal in I, we construct a literal pattern, which we call a "beam ma-

[3] *APX*-hardness for MINIMUM CONVEX COVER for input polygons without holes implies *APX*-hardness for the same problem for input polygons with holes.

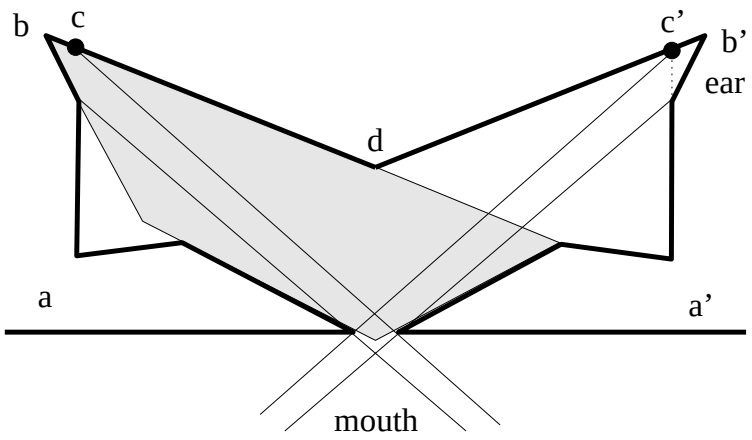


FIG. 7. *The beam machine.*

chine," as illustrated in Figure 7. A beam machine allows us to send a beam, i.e., a slim convex polygon in one of two possible directions out of the beam machine toward a structure that represents a clause. The beam machines of all literals of a variable are then combined into a variable structure, as illustrated in Figure 8. All these variable structures are then arranged in a half-circle such that the beams emitted from the beam machines reach the appropriate clause checkers, which are simple dents. An overview of the whole structure is given in an example in Figure 9. After this overview, let us give a more detailed description.

The beam machine that is constructed for each literal is shown in Figure 7. Since no two of the four vertices a, a', b, and b' see each other, at least four convex polygons are needed to cover the beam machine. Two of these are the maximal convex polygons a, c, d and a', c', d. The remaining areas around the mouth and the ear (the triangle) at b or b' can be covered by a large convex polygon shown in light gray in Figure 7. Finally, a fourth convex polygon is needed to cover the other ear (at b' in Figure 7). This polygon, which we call a beam, is very slim and can be extended indefinitely beyond the mouth outside the beam machine. The large light gray convex polygon thus acts as a switch: depending on whether we let it cover the ear at b or b', we can turn on the indefinite beam polygon at the other ear. However, we cannot turn on both beams and still use only four polygons to cover the beam machine. Note that we can "focus" and "aim" the beam by slightly bending the whole beam machine or by making the ears smaller.

The variable structure is illustrated in Figure 8. Its basic shape is butterfly-like. The beam machines for each occurrence of the variable in a literal in a clause are set on top of the butterfly with the positive literals on the right wing and the negative literals on the left wing of the butterfly. For each literal, we have a dent on the bottom line of the wing. If we cover each dent of the left or right wing with a maximal convex polygon, i.e., with a polygon that covers the whole dent and then extends canonically, then we have covered almost all of the left or right wing except for the area around the mouth of the variable structure and except for a small triangular region for each literal that lies between two dents. These triangles are called beam locks. We can cover the beam locks either by beams emanating from the beam machines or by a single large convex polygon which also covers the region around the mouth of the variable structure. Such a polygon is drawn in light gray in Figure 8. In a similar

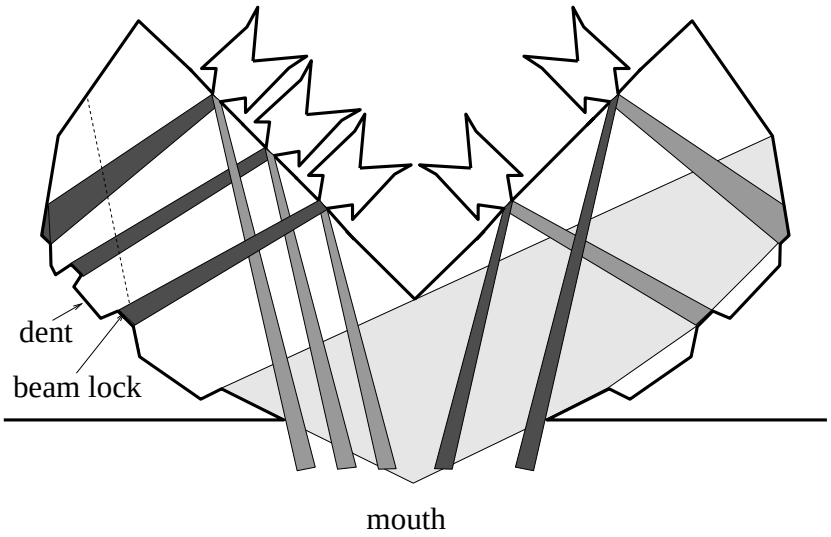


FIG. 8. *The variable structure.*

way as in the beam machine, this large convex polygon acts as a switch: in order to cover the whole variable structure with a minimum number of convex polygons, we can have the beam locks of only one wing covered with such a single polygon; the beam locks of the other wing must be covered by the beams of the beam machines. In Figure 8, beams that are turned on are drawn in dark gray, while beams that are turned off are medium gray. Thus, in Figure 8, all beam machines of positive literals are turned off, and all beam machines of negative literals are turned on and can shine infinitely far beyond the mouth of the variable structure.[4]

We need four convex polygons to cover each beam machine; thus we need $4l_i$ convex polygons to cover the beam machines in the variable structure for variable x_i. For each literal, we need an additional polygon to cover the dent, and we need one additional large switcher polygon to cover the mouth and the beam locks of either the positive or negative literals. Thus a minimum number of $5l_i + 1$ convex polygons are required to cover the variable structure of variable x_i. Note that if the beams of only one negative and one positive literal that are both aimed toward and beyond the mouth of the variable structure are turned on, then $5l_i + 2$ convex polygons are needed to cover the variable structure. On the other hand, if the beams of all (positive and negative) literals that cover the beam locks are turned on, there are still $5l_i + 1$ convex polygons needed to cover the variable structure, since we also need to cover the area around the mouth.

We arrange all variable structures in a half-circle-like shape above a base line, which contains triangular dents that represent the clauses, as illustrated in Figure 9. This is done in such a way that a beam emanating from a beam machine of a literal that appears in a clause reaches the corresponding dent (the clause checker) that represents that clause and thus covers it. Note that we can arrange the variable structures in such a way that they cannot interfere with each other; i.e., no convex polygon can cover any beam locks or areas around the mouth of two different variable structures. We can achieve this by making the angles at the mouth of each variable

[4]The beam machines have not been drawn exactly to scale in Figure 8.

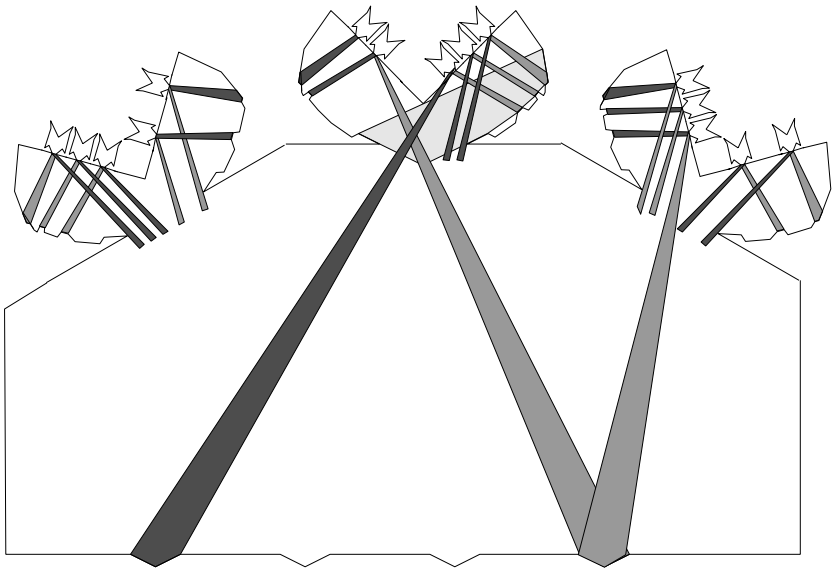


FIG. 9. *Overview of the construction.*

structure very small.

THEOREM 4. *Let I be an instance of* MAXIMUM 5-OCCURRENCE-3-SAT *consisting of n variables and m clauses with a total of l literals, and let I' be the corresponding instance of* MINIMUM CONVEX COVER. *Let OPT be the maximum number of satisfied clauses of I by any assignment of the variables. Let OPT' be the minimum number of convex polygons needed to cover the polygon of I', and let $\epsilon > 0$ be constant. Then*

$$OPT = m \Longrightarrow OPT' = 5l + n + 1,$$
$$OPT < (1 - 15\epsilon)m \Longrightarrow OPT' > 5l + n + 1 + \epsilon n.$$

Proof. The first implication is trivial: if we have a variable assignment that satisfies all variables, we turn on the beams that are aimed toward the clause checkers of all beam machines that represent literals that are satisfied by the assignment. We turn on the beams that are aimed toward the beam locks for all other beam machines. Thus we need $5l_i + 1$ convex polygons to cover the variable structure x_i. If we sum this up over all n variables, we obtain $5l + n$ convex polygons. We need one additional polygon to cover the space between the base line and the variable structures.

Since each clause is satisfied, we must have for each clause checker at least one beam turned on that covers it. Thus the convex polygons as just described cover all of I'.

We prove the second implication by proving its contraposition, i.e., $OPT' \leq 5l + n + 1 + \epsilon n \Longrightarrow OPT \geq (1 - 15\epsilon)m$. To this end, we show how to transform the convex polygons of any solution S' of the MINIMUM CONVEX COVER instance I' in such a way that their total number does not increase and in such a way that a truth assignment of the variables satisfying the desired number of clauses can be "inferred" from the convex polygons.

Suppose we are given a solution S' of the CONVEX COVER instance with $|S'| \leq 5l + n + 1 + \epsilon n$.

By construction, the variable generator for variable x_i must be covered by at least $5l_i+1$ convex polygons. Moreover, by construction, there is no convex polygon, which simultaneously covers a part of a beam lock in any variable generator and a part of a clause checker. There is not even a convex polygon which covers a part of a beam lock and touches the horizontal line, on which the clause checkers lie. Similarly, note that there is no convex polygon which can simultaneously cover a part of an ear of a beam machine and a part of any clause checker, except for the clause checker associated with the beam machine.

Proceed in the following order:

1. Determine which convex polygon in S' covers the midpoint on the line segment between the clause checkers of clause c_1 and c_2. Transform this polygon in such a way that it covers all of the area between the clause checkers and the variable generators. Note that no convex polygon that covers this midpoint can also cover any beam lock, ear of a beam machine, or clause checker. Therefore, we have a feasible solution after this step.
2. For each clause checker, proceed as follows: for each convex polygon in S' that covers part of the clause checker and that is not a regular beam which leads to a beam machine associated with the clause checker, turn the polygon into a beam to any of the associated beam machines.
3. If there exists a convex polygon in S' that covers parts of the interior of at least two different variable structures, then choose any variable structure in which it lies, and cut off all other parts. This operation results in a feasible solution since, by construction, such a polygon cannot cover the beam locks or the area around the mouths of two different variable structures.
4. For each variable structure, proceed as follows:
   - If the variable structure for x_i is covered by $5l_i+2$ or more convex polygons, then rearrange the convex polygons in such a way that all beams that point to clause checkers are turned on for positive and negative literals. By construction, this is always possible with $5l_i + 2$ convex polygons.
   - If the variable structure for x_i is covered by $5l_i + 1$ convex polygons and one beam from a beam machine for literal x_i ($\neg x_i$) that is aimed at its associated clause checker is turned on, then rearrange all convex polygons in the variable generator in such a way that all beams from beam machines for literal x_i ($\neg x_i$) that are aimed at the associated clause checkers are turned on.

The convex cover obtained this way is still a feasible solution. After this transformation, we have for each variable structure x_i one of the following cases:

- for all negative and positive literals, the beams that are aimed toward the clause checkers are turned on;
- only for all positive or negative literals, the beams that are aimed toward the clause checkers are turned on;
- for negative and positive literals, the beams that are aimed toward the beam locks are turned on.

We set the truth values for the variables as follows: if all beams of literal x_i ($\neg x_i$) that are aimed at clause checkers and no beams of literal $\neg x_i$ (x_i) that are aimed at clause checkers are turned on, then let the variable x_i have truth value TRUE (FALSE). If either all or no beams (of both literals x_i and $\neg x_i$) that are aimed at clause checkers are turned on, then let variable x_i be TRUE.

By construction, every solution of I' must consist of at least $5l + n + 1$ convex polygons. If we transform a solution of I' with $5l + n + 1 + \epsilon n$ convex polygons as indicated above, we get at most ϵn variable structures in which the beams of all literals (positive and negative) that are aimed at the clause checkers are turned on. By assigning all these variables the value TRUE, we falsify at most five clauses for each variable, since each variable appears at most five times as a literal.

Therefore, we get a solution of I with at least $m - 5\epsilon n$ clauses satisfied. Since $3m \geq n$, the solution has at least $m(1 - 15\epsilon)$ satisfied clauses. □

In the so-called promise problem [1] of MAXIMUM 5-OCCURRENCE-3-SAT as described above, we are promised that either all clauses are satisfiable or at most a fraction of $1 - 15\epsilon$ of the clauses is satisfiable, and we are to find out which of the two possibilities is true. This problem is *NP*-hard for sufficiently small values of $\epsilon > 0$ (see [1]). Therefore, Theorem 4 implies that the promise problem for MINIMUM CONVEX COVER, where we are promised that the minimum solution contains either $5l + n + 1$ convex polygons or at least $5l + n + 1 + \epsilon n$ convex polygons, is *NP*-hard as well for sufficiently small values of $\epsilon > 0$. Therefore, MINIMUM CONVEX COVER cannot be approximated with a ratio of $\frac{5l+n+1+\epsilon n}{5l+n+1} \geq 1 + \frac{\epsilon n}{25n+n+1} \geq 1 + \frac{\epsilon}{27}$, where we have used that $l \leq 5n$ and $n \geq 1$. This establishes the following theorem.

THEOREM 5. MINIMUM CONVEX COVER *on input polygons with or without holes is APX-hard.*

7. Conclusion. We have proposed a polynomial-time approximation algorithm for MINIMUM CONVEX COVER that achieves an approximation ratio that is logarithmic in the number of vertices of the input polygon. This has been achieved by showing that there is a discretized version of the problem using no more than three times the number of cover polygons. The discretization may be a first step toward answering the long-standing open question of whether the decision version of the MINIMUM CONVEX COVER problem is in *NP* [18]: we know now that there always exists an optimum solution such that the convex polygons in such an optimum solution contain only a polynomial number of vertices and that a considerable fraction of these vertices are actually vertices from the input polygon; however, all other vertices of the convex polygons could still need a superpolynomial number of bits for their coordinates to be expressed. Apart from the discretization, our algorithm applies a MINIMUM SET COVER approximation algorithm to a MINIMUM SET COVER instance with an exponential number of sets that are represented only implicitly, through the geometry. We propose an algorithm that picks the best of the implicitly represented sets with a dynamic programming approach and hence runs in polynomial time. This technique may prove to be of interest for other problems as well. Moreover, by showing *APX*-hardness, we have eliminated the possibility of the existence of a polynomial-time approximation scheme for this problem. However, polynomial-time algorithms could still achieve constant approximation ratios. Whether our algorithm is the best asymptotically possible is therefore an open problem. Furthermore, our algorithm has a rather excessive running time of $O(n^{29} \log n)$, and it is by no means clear how this can be improved substantially.

Acknowledgments. We would like to thank anonymous referees for numerous valuable comments that greatly improved this paper; in particular, we thank the referees for pointing us to [11] and [16] and for providing the running time analysis in section 5 that reduces the overall running time of our algorithm from $O(n^{44})$, which we had in earlier versions of this paper, to $O(n^{29} \log n)$.

REFERENCES

[1] S. Arora and C. Lund, *Hardness of approximations*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS Publishing Company, Boston, 1996, pp. 399–446.

[2] B. Chazelle and D. P. Dobkin, *Optimal convex decompositions*, in Computational Geometry, Mach. Intelligence Pattern Recogn. 2, North–Holland, Amsterdam, 1985, pp. 63–133.

[3] P. Crescenzi and V. Kann, *A compendium of NP optimization problems*, in Complexity and Approximation. Combinatorial Optimization Problems and Their Approximability Properties, G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, eds., Springer-Verlag, Berlin, 1999, pp. 87–122.

[4] J. C. Culberson and R. A. Reckhow, *Covering polygons is hard*, J. Algorithms, 17 (1994), pp. 2–44.

[5] D. P. Dobkin, H. Edelsbrunner, and M. H. Overmars, *Searching for empty convex polygons*, Algorithmica, 5 (1990), pp. 561–571.

[6] S. Eidenbenz, C. Stamm, and P. Widmayer, *Inapproximability results for guarding polygons and terrains*, Algorithmica, 31 (2001), pp. 79–113.

[7] S. Eidenbenz, *(In-)Approximability of Visibility Problems on Polygons and Terrains*, Ph.D. thesis, Dissertation ETH 13683, Zürich, Switzerland, 2000.

[8] S. Eidenbenz and P. Widmayer, *An approximation algorithm for minimum convex cover with logarithmic performance guarantee*, in Algorithms—ESA 2001 (Århus), Lecture Notes in Comput. Sci. 2161, Springer-Verlag, Berlin, 2001, pp. 333–343.

[9] D. S. Franzblau, *Performance guarantees on a sweep-line heuristic for covering rectilinear polygons with rectangles*, SIAM J. Discrete Math., 2 (1989), pp. 307–321.

[10] S. Ghosh, *Approximation algorithm for art gallery problems*, in Proceedings of the Canadian Information Processing Society Congress, Mississauga, ON, Canada, 1987.

[11] J. Gudmundsson and C. Levcopoulos, *Close approximations of minimum rectangular coverings*, J. Comb. Optim., 3 (1999), pp. 437–452.

[12] D. H. Green, *The decomposition of polygons into convex parts*, in Computational Geometry, Advances in Computer Research 1, F. P. Preparata, ed., JAI Press, London, 1983, pp. 235–259.

[13] D. Hochbaum, *Approximating covering and packing problems: Set cover, vertex cover, independent set, and related problems*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS Publishing Company, Boston, 1996, pp. 94–143.

[14] V. S. A. Kumar and H. Ramesh, *Covering rectilinear polygons with axis-parallel rectangles*, in Proceedings of the 31st ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 445–454.

[15] A. Lingas, *The power of nonrectilinear holes*, in Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 140, Springer-Verlag, Berlin, 1982, pp. 369–383.

[16] A. Lingas and V. Soltan, *Minimum convex partition of a polygon with holes by cuts in given directions*, Theory Comput. Syst., 31 (1998), pp. 507–538.

[17] J. O'Rourke and K. J. Supowit, *Some NP-hard polygon decomposition problems*, IEEE Trans. Inform. Theory, 29 (1983), pp. 181–190.

[18] T. Shermer, *Recent results in art galleries*, Proc. IEEE, 80 (1992), pp. 1384–1399.

[19] J. Urrutia, *Art gallery and illumination problems*, in Handbook on Computational Geometry, J.-R. Sack and J. Urrutia, eds., Elsevier, New York, 2000, pp. 973–1027.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 671–680

PARALLEL PROCESSOR SCHEDULING WITH LIMITED NUMBER OF PREEMPTIONS*

OLIVER BRAUN† AND GÜNTER SCHMIDT†

Abstract. In this paper, we compare the makespan of preemptive and i-preemptive schedules where only a limited number i of preemptions is allowed. The problem is to schedule n independent jobs on m identical processors that operate in parallel. The objective is to minimize the makespan, i.e., the completion time of the last job that finishes. We show that the ratio of the optimal i-preemptive schedule length $C_{max}^{ip^*}$ versus the optimal preemptive schedule length $C_{max}^{p^*}$ is bounded from above by $C_{max}^{ip^*} \leq (2 - 2/(m/(i+1)+1))C_{max}^{p^*}$. Furthermore, we show that the ratio of the length C_{max}^{LPT} of a nonpreemptive schedule following the *longest processing time (LPT)* rule versus the optimal preemptive schedule length $C_{max}^{p^*}$ is bounded from above by exactly the same bound when $i = 0$.

Key words. parallel processor scheduling, preemptive scheduling, nonpreemptive scheduling, i-preemptive scheduling, longest processing time rule, schedule length, worst-case analysis

AMS subject classifications. 90B35, 68R05, 68M20

PII. S0097539702410697

1. Introduction.

In this paper, we compare the makespan of preemptive, i-preemptive, and LPT schedules for the problem in which a set of n independent jobs has to be scheduled on m identical processors that operate in parallel. The objective is to minimize the makespan, i.e., the maximum completion time of any job. Using the notation by Braun [2], we mean by an i-preemptive schedule that the maximum number of preemptions is bounded from above by a nonnegative integer number i. A preemptive schedule is allowed to interrupt a job and later resume its execution without any loss of time. Also, the minimum time slice for preempting a job may be arbitrarily small. In nonpreemptive schedules, a job is started and executed to completion without any interruption. It is easy to construct an optimal preemptive schedule with McNaughton's *wrap around* rule [9], whereas it is an NP-hard problem to construct an optimal nonpreemptive schedule. The *longest processing time (LPT)* rule, which orders the jobs in nondecreasing order of their processing times, is a $(4/3 - 1/3m)$-approximation algorithm for the nonpreemptive case [6]. Preemptive, nonpreemptive, and LPT schedules are commonly studied in the literature. An analysis of the complexity of these problems, along with several of their variants and special cases, can be found, e.g., in [1], [5], [3], and [10].

It is easy to see that an optimal preemptive schedule is never longer than an optimal nonpreemptive one. The time savings of a preemptive schedule are bought at the costs of moving a job off a processor before it is finished and at the costs of saving information about the job while it is waiting to be resumed. The costs of preempting a job may be neglected in a computer environment. Here, the costs of preempting the execution of a job in the main memory and transferring it to and from the virtual memory, such as mass storage devices, may be ignored. Generally, however, there arise costs with job preemptions such as inventory and transportation

*Received by the editors October 16, 2002; accepted for publication (in revised form) January 3, 2003; published electronically April 17, 2003.
http://www.siam.org/journals/sicomp/32-3/41069.html
†Department of Information and Technology Management, Saarland University, 66041 Saarbrücken, Germany (ob@itm.uni-sb.de, gs@itm.uni-sb.de).

costs. A small number of preemptions, in general, has advantages such as lower work-in-process inventories, reduced material handling costs, fewer tooling changes, quality improvements, and simplified production planning of materials and labor.

We investigate the following question: How much more effective (with respect to the objective of minimizing the makespan) can optimal preemptive scheduling be compared to optimal i-preemptive scheduling? More precisely, over all instances I, what is the least upper bound on $C_{max}^{ip^*}(I)/C_{max}^{p^*}(I)$, where $C_{max}^{ip^*}$ and $C_{max}^{p^*}$ denote an optimal i-preemptive and an optimal preemptive makespan, respectively? For a job system with arbitrary precedence constraints, Liu conjectured in [8] that the ratio of the optimal nonpreemptive makespan versus the optimal preemptive makespan is bounded above by $2 - 2/(m+1)$. His upper bound proof was found to be incorrect. Coffman and Garey [4] proved Liu's conjecture for a system with arbitrary precedence constraints and two processors. Hong and Leung [7] showed the correctness of the bound for unit execution time (UET) and also for tree-structured job systems. Their proof for tree-structured job systems is also valid for independent jobs.

The paper is organized as follows. In section 2, we show that the ratio of the optimal i-preemptive makespan $C_{max}^{ip^*}$ versus the optimal preemptive makespan $C_{max}^{p^*}$ is bounded from above by $C_{max}^{ip^*} \leq (2 - 2/(m/(i+1)+1))C_{max}^{p^*}$ for $0 \leq i \leq m-1$. It follows from [9] that no more than $m-1$ preemptions are necessary to construct a schedule which is optimal concerning the makespan. Then we show that the ratio of the makespan C_{max}^{LPT} of a nonpreemptive schedule following the LPT rule versus the optimal preemptive makespan $C_{max}^{p^*}$ is bounded from above by $C_{max}^{LPT} \leq (2m/(m+1))C_{max}^{p^*}$. For each of the two bounds, we give an example which shows the tightness of the bounds. Concluding remarks are given in section 3.

2. Worst-case analysis. The scheduling problem under consideration is as follows. There are n jobs $J_1, \ldots, J_n$ with processing times $p_1, \ldots, p_n$ that have to be scheduled on m identical parallel processors $P_1, \ldots, P_m$ so as to minimize the makespan C_{max}. C_{max} is the maximum of all finish times $C_j, j = 1, \ldots, n$, of all jobs. Each processor may work only on one job at a time, and each job may be processed by only one machine at a time. $C_{max}^{p^*}$ is the makespan (schedule length) of an optimal preemptive schedule, $C_{max}^{ip^*}$ is the makespan of an optimal i-preemptive schedule, and C_{max}^{LPT} is the makespan of a schedule constructed with the LPT rule. The LPT rule works in the following way: First, sort the jobs in a list in order of nonincreasing processing times. Then assign the first unscheduled job in the list to any one of the least loaded processors. An optimal i-preemptive schedule S allows at most i preemptions. We assume that preemptions are allowed only on the i processors $P_2, \ldots, P_{i+1}$. With $p_{max} = \max_{j=1,\ldots,n} p_j$ we denote the maximum processing time of a job, and with $\sum p_j = \sum_{j=1}^{n} p_j$ we denote the sum of all processing times.

McNaughton [9] has given a lower bound for the makespan of preemptive schedules:

$$C_{max}^{p^*} = \max\left\{ \frac{\sum p_j}{m} \ , \ p_{max} \right\}. \tag{2.1}$$

The *wrap around* rule of McNaughton constructs optimal schedules with at most $m-1$ preemptions. If at least $m-1$ preemptions are allowed, then one always can construct an optimal schedule with McNaughton's algorithm. Therefore, we have to investigate for i-preemptive scheduling only the case when $0 \leq i \leq m-1$.

Let J_k be a job with processing time p_k and completion time $C_k = C_{max}^{ip^*}$. We assume without loss of generality that job J_k is processed on processor P_{i+1} or on

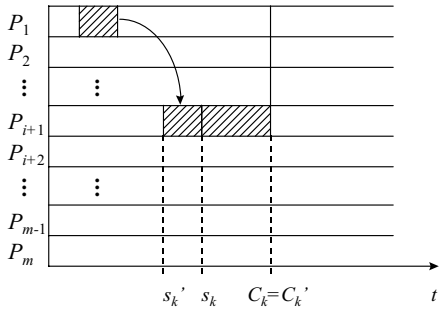


FIG. 2.1. *Lemma* 2.1.

processor P_{i+2}. The structure of the proof is as follows. If $p_k \geq \sum p_j/m$, we have $C_{max}^{ip^*} = C_{max}^{p^*}$, as by (2.1) a preemptive schedule cannot be shorter than p_k. So we assume that $p_k < \sum p_j/m$. In Lemma 2.1, we show that in an optimal i-preemptive schedule S, there always exists a job J_k with processing time p_k and completion time $C_k = C_{max}^{ip^*}$ which is not interrupted. Note that preemptions are allowed only on the i processors $P_2, \ldots, P_{i+1}$. In Lemma 2.2, we prove that the start time of J_k is not larger than $(\sum p_j - (i+1)p_k)/m$. In Lemmas 2.3 and 2.4, we show that in the case when $p_k \leq (m/(m+i+1))C_{max}^{p^*}$ and $p_k > (m/(m+i+1))C_{max}^{p^*}$, the inequality $C_{max}^{ip^*} \leq (2 - 2/(m/(i+1)+1))C_{max}^{p^*}$ is fulfilled. Next we prove the lemmas.

LEMMA 2.1. *Let J_k be a job with processing time $p_k < \sum p_j/m$ and completion time $C_k = C_{max}^{ip^*}$. In an optimal i-preemptive schedule S, J_k is not preempted.*

Proof. This proof is by contradiction. Note that a job is preempted one time at most and that the first scheduled part of job J_k must be scheduled on one of the first $i+1$ processors, because only on these processors are preemptions allowed. If in schedule S job J_k would be preempted, it would be possible to construct a new schedule S' with the same makespan and with J_k not preempted as follows. Without loss of generality, let P_1 be the processor where J_k is preempted in schedule S, and let P_{i+1} be the processor where the processing of J_k is finished. If we move the part of job J_k that is scheduled on processor P_1 immediately before the start time s_k of J_k on processor P_{i+1} and schedule the load of $P_1, \ldots, P_{i+1}$ with McNaughton's rule, we obtain a new schedule S' with the following properties (see Figure 2.1).

1. The finish times of processors $P_{i+2}, \ldots, P_m$ remain unchanged.
2. The total capacity to be scheduled on processors $P_1, \ldots, P_{i+1}$ remains unchanged in schedule S'. As it is allowed to schedule this capacity with McNaughton's rule, and as there is no job J_j with $p_j = C_{max}^{ip^*}$, the finish times of processors $P_2, \ldots, P_{i+1}$ in schedule S' are not larger than C_k of schedule S.

Thus schedule S' has the same makespan as schedule S, and J_k is not preempted. □

LEMMA 2.2. *Let J_k be a job with processing time $p_k < \sum p_j/m$ and completion time $C_k = C_{max}^{ip^*}$. For $0 \leq i \leq m-1$, the maximum number of allowed preemptions, the start time s_k of J_k in an optimal i-preemptive schedule S is not larger than $(\sum p_j - (i+1)p_k)/m$.*

Proof. In Lemma 2.1, it has been shown that in an optimal i-preemptive schedule S, job J_k with $C_k = C_{max}^{ip^*}$ is not preempted. Note that only on the i processors $P_2, \ldots, P_{i+1}$ are preemptions allowed. Thus we distinguish two cases.

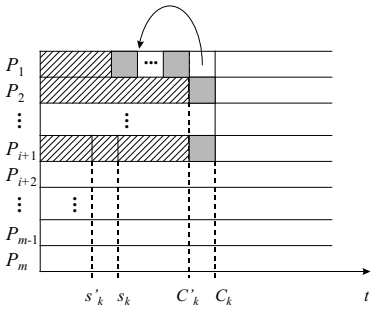


FIG. 2.2. *Lemma* 2.2, *Case* 1a.

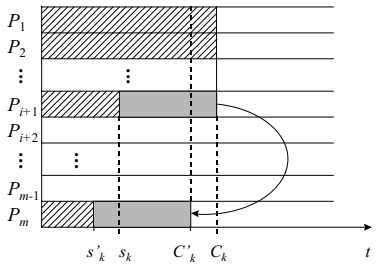


FIG. 2.3. *Lemma* 2.2, *Case* 1b.

Case 1. In an optimal i-preemptive schedule S, J_k is processed on one of the first $i+1$ processors (without loss of generality on P_{i+1}).

Note that we are allowed to schedule the load on the first $i+1$ processors with McNaughton's *wrap around* rule and that there is no job J_j with $p_j = C_{max}^{ip^*}$. In an optimal i-preemptive schedule S, the load on the first $i+1$ processors is at least $(i+1)C_k$. If not, one could construct a new schedule S' with a smaller makespan as follows. Without loss of generality, let P_1 be a processor with load less than C_k. By scheduling the load on the first $i+1$ processors with McNaughton's rule, we would get a new schedule S' with the following properties (see Figure 2.2).

1. The total capacity to be scheduled on processors $P_1, \ldots, P_{i+1}$ remains unchanged in schedule S'. As we are allowed to schedule this capacity with McNaughton's rule, and as there is no job J_j with $p_j = C_{max}^{ip^*}$, the finish times of processors $P_1, \ldots, P_{i+1}$ in schedule S' are less than C_k of schedule S, and the number of preemptions is still at most i.
2. The finish times of processors $P_{i+2}, \ldots, P_m$ remain unchanged.

Thus schedule S' would have a smaller makespan than schedule S, and we have proved that the load on the first $i+1$ processors is at least $(i+1)C_k$.

Next, we show that the last $m-(i+1)$ processors have a load of at least $(m-(i+1))s_k$. If not, one could construct a new schedule S' with smaller makespan as follows. Without loss of generality, let P_m be a processor with load less than s_k. If we would move job J_k to processor P_m, we would obtain a new schedule S' with the following properties (see Figure 2.3).

1. The total capacity to be scheduled on processors $P_1, \ldots, P_{i+1}$ reduces by p_k in schedule S'. As we are allowed to schedule this capacity with McNaughton's *wrap around* rule, and as there is no job J_j with $p_j = C_{max}^{ip^*}$, the finish times of processors $P_1, \ldots, P_{i+1}$ in schedule S' are less than C_k of schedule S, and the number of preemptions is still at most i.
2. The finish times of processors $P_{i+2}, \ldots, P_{m-1}$ remain unchanged.
3. As the finish time on P_m was less than s_k (in schedule S), the finish time on P_m in schedule S' is less than C_k of schedule S.

Thus schedule S' would have a smaller makespan than schedule S, and we have proved that the load on the last $m-(i+1)$ processors is $(m-(i+1))s_k$.

In total, we have

$$\sum p_j \geq (i+1)C_k \quad + \quad (m-(i+1))s_k = (i+1)p_k + ms_k$$

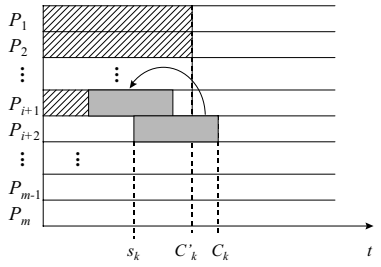


FIG. 2.4. *Lemma* 2.2, *Case* 2a.

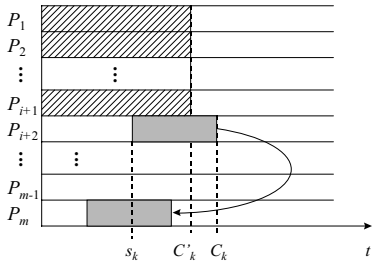


FIG. 2.5. *Lemma* 2.2, *Case* 2b.

or, equivalently,

$$s_k \leq \frac{\sum p_j - (i+1)p_k}{m}.$$

Case 2. In an optimal i-preemptive schedule S, J_k is processed on one of the last $m-(i+1)$ processors (without loss of generality on P_{i+2}).

The first $i+1$ processors have a load of at least $(i+1)C_k - p_k$. If not, we could generate a new schedule S' with a smaller makespan than S as follows. We are allowed to schedule the load on the first $i+1$ processors with McNaughton's rule, and there is no job J_j with processing time $p_j = C^{ip^*}_{max}$. Thus, in schedule S, there must be a processor (without loss of generality P_{i+1}) with load less than $C_k - p_k = s_k$. If we move job J_k to processor P_{i+1}, we would obtain a new schedule S' with the following properties (see Figure 2.4).

1. The finish times of processors $P_1, \ldots, P_i$ remain unchanged.
2. As the finish time on P_{i+1} was less than s_k (in schedule S), the finish time on P_{i+1} in schedule S' is less than C_k of schedule S.
3. The finish time of processor P_{i+2} reduces by p_k.
4. The finish times of processors $P_{i+3}, \ldots, P_m$ remain unchanged.

Thus schedule S' would have a smaller makespan than schedule S, and we have proved that the load on the first $i+1$ processors is at least $(i+1)C_k - p_k$.

Next we prove that the last $m-(i+1)$ processors have a load of at least $(m-(i+1))s_k + p_k$. If not, we could generate a new schedule S' with a smaller makespan than S as follows. Without loss of generality, let P_m be a processor with load less than s_k. If we would move job J_k to processor P_m, we would obtain a new schedule S' with the following properties (see Figure 2.5).

1. The finish times of processors $P_1, \ldots, P_{i+1}$ remain unchanged.
2. The finish time of processor P_{i+2} reduces by p_k.
3. The finish times of processors $P_{i+3}, \ldots, P_{m-1}$ remain unchanged.
4. As the finish time on P_m was less than s_k (in schedule S), the finish time on P_m in schedule S' is less than C_k of schedule S.

Thus schedule S' would have a smaller makespan than schedule S, and we have proved that the load on the last $m-(i+1)$ processors is $(m-(i+1))s_k + p_k$.

In total, we have

$$\begin{aligned}\sum p_j &\geq (i+1)C_k - p_k \quad + \quad (m-(i+1))s_k + p_k \\ &= (i+1)(s_k+p_k) - p_k \quad + \quad (m-(i+1))s_k + p_k = (i+1)p_k + ms_k\end{aligned}$$

or, equivalently,

$$s_k \leq \frac{\sum p_j - (i+1)p_k}{m}. \qquad \square$$

LEMMA 2.3. *Let S be an optimal i-preemptive schedule with makespan $C_{max}^{ip^*}$. Let J_k be a job with $C_k = C_{max}^{ip^*}$, $p_k < \sum p_j/m$, and*

$$p_k \leq C_{max}^{p^*} \left(\frac{m}{m+i+1} \right). \tag{2.2}$$

For $0 \leq i \leq m-1$, the maximum number of allowed preemptions, we have, for all instances of the problem $P \mid i - pmtn \mid C_{max}$,

$$C_{max}^{ip^*} \leq \left(2 - \frac{2}{\left(\frac{m}{i+1}\right)+1} \right) C_{max}^{p^*}.$$

Proof. From Lemma 2.2 we know that job J_k is not preempted. Thus we have for the schedule length of the optimal i-preemptive schedule

$$C_{max}^{ip^*} = s_k + p_k.$$

With the help of Lemma 2.2, we have

$$C_{max}^{ip^*} \leq \frac{\sum p_j - (i+1)p_k}{m} + p_k = \frac{\sum p_j}{m} + \frac{m-(i+1)}{m} p_k.$$

McNaughton's bound (2.1) and assumption (2.2) lead to

$$C_{max}^{ip^*} \leq C_{max}^{p^*} + \frac{m-(i+1)}{m} \frac{m}{m+(i+1)} C_{max}^{p^*} = \left(2 - \frac{2}{(\frac{m}{i+1})+1} \right) C_{max}^{p^*}. \qquad \square$$

LEMMA 2.4. *Let S be an optimal i-preemptive schedule with makespan $C_{max}^{ip^*}$. Let J_k be a job with $C_k = C_{max}^{ip^*}$, $p_k < \sum p_j/m$, and*

$$p_k > C_{max}^{p^*} \left(\frac{m}{m+i+1} \right). \tag{2.3}$$

For $0 \leq i \leq m-1$, the maximum number of allowed preemptions, we have, for all instances of the problem $P \mid i - pmtn \mid C_{max}$,

$$C_{max}^{ip^*} \leq \left(2 - \frac{2}{\left(\frac{m}{i+1}\right)+1} \right) C_{max}^{p^*}.$$

Proof. This proof is by contradiction. We assume that there is an optimal i-preemptive schedule S with $C_{max}^{ip^*} > (2-2/(m/(i+1)+1))C_{max}^{p^*}$. In this case, there is always a processor with a load of at most $(m/(m+i+1))C_{max}^{p^*}$, as we can see as follows. If there would not exist a processor with a load of maximum $(m/(m+i+1))C_{max}^{p^*}$,

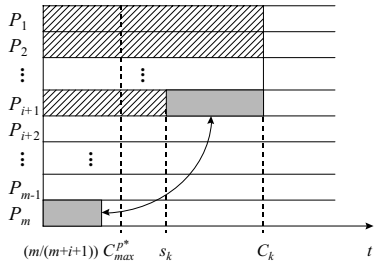


FIG. 2.6. *Lemma* 2.4, *Case* 1.

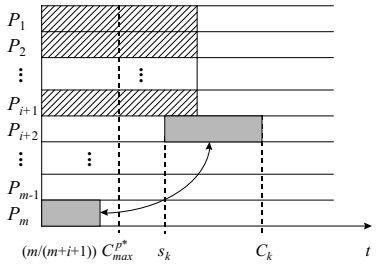


FIG. 2.7. *Lemma* 2.4, *Case* 2.

we would have

$$\begin{aligned}\sum p_j &> (i+1)C^{ip^*}_{max} \quad + \quad (m-(i+1))\left(\frac{m}{m+i+1}\right)C^{p^*}_{max} \\ &> (i+1)\left(2-\frac{2}{\left(\frac{m}{i+1}\right)+1}\right)C^{p^*}_{max} \quad + \quad (m-(i+1))\left(\frac{m}{m+i+1}\right)C^{p^*}_{max} \\ &= mC^{p^*}_{max}\ .\end{aligned}$$

This would be a contradiction to McNaughton's bound (2.1). Thus there is a processor with a load of at most $m/(m+i+1)C^{p^*}_{max}$. Without loss of generality, we assume that this processor is P_m with finish time F_m.

Case 1. In an optimal i-preemptive schedule S, J_k is processed on one of the first $i+1$ processors (without loss of generality on P_{i+1}). In this case, one could construct a new schedule S' with smaller makespan as follows. If we would switch job J_k with the jobs processed on processor P_m, we would obtain a new schedule S' with the following properties (see Figure 2.6).

1. Because of (2.3), we know that $p_k > (m/(m+i+1))C^{p^*}_{max}$. By switching all the jobs scheduled on P_m with J_k, the total capacity to be scheduled on processors $P_1, \ldots, P_{i+1}$ reduces by $p_k - F_m$ in schedule S'. As we are allowed to schedule the remaining capacity with McNaughton's *wrap around* rule, and as there is no job J_j with $p_j = C^{ip^*}_{max}$, the finish times of processors $P_1, \ldots, P_{i+1}$ in schedule S' are less than C_k of schedule S.
2. The finish times of processors $P_{i+2}, \ldots, P_{m-1}$ remain unchanged.
3. As the finish time F_m on P_m was less than p_k (in schedule S), and as J_k was not the only job scheduled on P_{i+1} (in schedule S), the finish time on P_m in schedule S' is less than C_k of schedule S.

Thus the makespan generated by S' is smaller than the makespan generated by S, which leads to a contradiction to the assumption of the optimality of S.

Case 2. In an optimal i-preemptive schedule S, J_k is processed on one of the last $m-(i+1)$ processors (without loss of generality on P_{i+2}). In this case, one could construct a new schedule S' with smaller makespan as follows. If we switch job J_k with the jobs processed on processor P_m, we obtain a new schedule S' with the following properties (see Figure 2.7).

1. The finish times of processors $P_1, \ldots, P_{i+1}$ remain unchanged.
2. Because of (2.3), we know that $p_k > (m/(m+i+1))C^{p^*}_{max}$. As the finish time F_m on P_m was less than p_k (in schedule S), the finish time on P_{i+2} in schedule S' is less than C_k of schedule S.

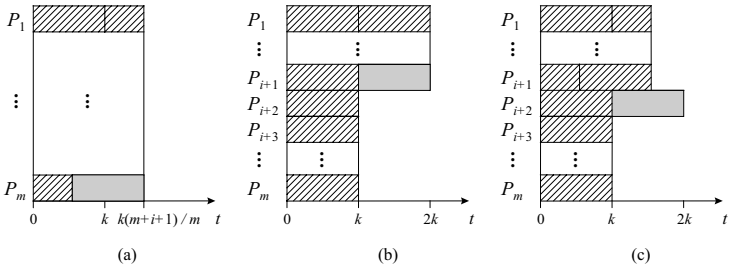


FIG. 2.8. *Optimal preemptive and i-preemptive schedules.*

3. The finish times of processors $P_{i+3}, \ldots, P_{m-1}$ remain unchanged.
4. As J_k was not the only job to be scheduled on P_{i+1} in schedule S, the finish time on P_m in schedule S' is less than C_k of schedule S.

Thus the makespan generated by S' is smaller than the makespan generated by S, which leads to a contradiction to the assumption of the optimality of S. □

THEOREM 2.5. *Let S be an optimal i-preemptive schedule with makespan $C_{max}^{ip^*}$. For $0 \leq i \leq m-1$, the maximum number of allowed preemptions, we have, for all instances of the problem $P \mid i - pmtn \mid C_{max}$,*

$$C_{max}^{ip^*} \leq \left(2 - \frac{2}{\left(\frac{m}{i+1}\right)+1}\right) C_{max}^{p^*} .$$

Proof. The proof is by Lemmas 2.3 and 2.4. □

To show the tightness of the bound $C_{max}^{ip^*} \leq (2 - 2/(m/(i+1)+1))C_{max}^{p^*}$, we consider the following problem instance: n jobs are to be scheduled on m processors with $n = m+i+1$ and $p_j = k, j = 1, \ldots, n$. The optimal i-preemptive schedule always has length $2k$, whereas the optimal preemptive schedule length is $k(m+i+1)/m$ (see Figure 2.8).

Job J_k with $C_k = C_{max}^{ip^*}$ is marked grey. In figure (a), the jobs are scheduled with McNaughton's rule. In figure (b), job J_k is scheduled on the first $i+1$ processors which allow preemptions (without loss of generality on P_{i+1}). In figure (c), job J_k is scheduled on the last $m-(i+1)$ processors which forbid preemptions (without loss of generality on P_{i+2}). In both cases (b) and (c), there must exist a processor that schedules two jobs without preemption.

We have

$$\frac{C_{max}^{ip^*}}{C_{max}^{p^*}} = \frac{2k}{k(m+i+1)/m} = \left(2 - \frac{2}{\left(\frac{m}{i+1}\right)+1}\right).$$

In the following, we show that the relation between a schedule constructed with the *LPT* rule and an optimal preemptive schedule is bounded from above by $2 - 2/(m+1)$.

THEOREM 2.6. *For schedules generated with the LPT rule, we have, for all instances,*

$$C_{max}^{LPT} \leq \left(2 - \frac{2}{m+1}\right) C_{max}^{p^*}.$$

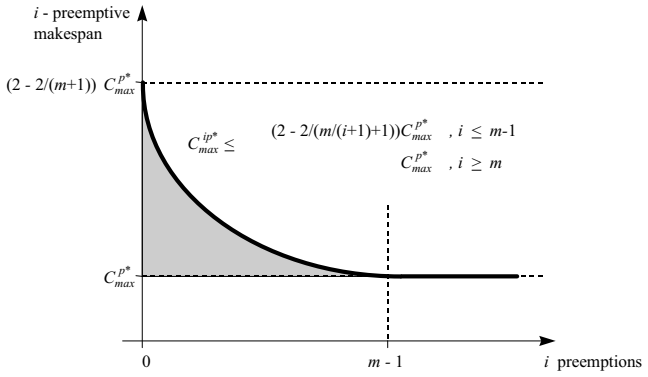


FIG. 3.1. *i-preemptive makespan (worst-case).*

Proof. Let J_k be the job that determines the schedule length of the LPT schedule. If $p_k \leq (m/(m+1))C_{max}^{p^*}$, we have

$$C_{max}^{LPT} = s_k + p_k \leq \frac{\sum p_j - p_k}{m} + p_k = \frac{\sum p_j}{m} + \left(1 - \frac{1}{m}\right) p_k$$
$$\leq C_{max}^{p^*} + \left(1 - \frac{1}{m}\right) C_{max}^{p^*} \left(\frac{m}{m+1}\right) = \left(2 - \frac{2}{m+1}\right) C_{max}^{p^*} .$$

If $p_k > (m/(m+1))C_{max}^{p^*}$, we have

$$p_k > C_{max}^{p^*} \left(\frac{m}{m+1}\right) \geq \left(\frac{\sum p_j}{m}\right) \left(\frac{m}{m+1}\right) = \frac{\sum p_j}{m+1}$$
$$\Longrightarrow p_k > \frac{\sum p_j, j \neq k}{m} \quad \geq \quad s_k$$
$$\Longrightarrow s_k < p_k.$$

s_k is only less than p_k if $s_k = 0$. (Note that the schedule is generated with the LPT rule.) Thus we have $C_{max}^{LPT} = C_{max}^{p^*}$ if $p_k > (m/(m+1))C_{max}^{p^*}$. □

For $i = 0$, this bound is also met by schedules generated by the LPT rule.

3. Summary. We compared the length of preemptive and i-preemptive schedules for the problem in which a set of n independent jobs has to be scheduled on m identical processors that operate in parallel. The objective is to minimize the makespan. We showed that the ratio of the optimal i-preemptive schedule length $C_{max}^{ip^*}$ (with a limited number $0 \leq i \leq m-1$ of preemptions) versus the optimal preemptive schedule length $C_{max}^{p^*}$ is bounded from above by $C_{max}^{ip^*} \leq (2 - 2/(m/(i+1)+1))C_{max}^{p^*}$. The grey region in Figure 3.1 displays all possible makespans as a function of the number i of maximum allowed preemptions.

Furthermore, we showed that the ratio of the length C_{max}^{LPT} of a nonpreemptive schedule generated by the LPT rule versus the optimal preemptive schedule length $C_{max}^{p^*}$ is bounded from above by exactly the same bound if $i = 0$. We gave an example which shows the tightness of the bounds.

Further research is needed to investigate the trade-off between the objectives of minimizing the makespan (time objective) and minimizing the number of preemptions (cost objective), i.e., to find a minimal length schedule for a given i. Clearly, if

$i \geq m-1$, one can generate an optimal schedule with the algorithm of McNaughton. For any value between 0 and $m-2$, however, the problem of finding a minimal length schedule is NP-hard.

The same problem arises if we try to find a schedule with a makespan not greater than $C_{max}^{allowed}$ and with a minimal number of preemptions. Clearly, if $C_{max}^{allowed} < C_{max}^{p^*}$, then there is no feasible schedule, and if $C_{max}^{allowed} \geq 2 - (2/(m+1))C_{max}^{p^*}$, then any LPT schedule meets this bound by generating no preemptions. For any value between these two values, however, the problem of minimizing the number of preemptions is NP-hard.

REFERENCES

[1] J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz, *Scheduling Computer and Manufacturing Processes*, 2nd ed., Springer-Verlag, Berlin, 2001.

[2] O. Braun, *Scheduling with Limited Available Processors and with Limited Number of Preemptions*, Ph.D. thesis, Saarland University, Saarbrücken, Germany, 2002 (in German).

[3] E. G. Coffman, Jr., ed., *Computer and Job-Shop Scheduling Theory*, John Wiley, New York, 1976.

[4] E. G. Coffman, Jr., and M. R. Garey, *Proof of the* 4/3 *conjecture for preemptive vs. nonpreemptive two-processor scheduling*, J. ACM, 20 (1993), pp. 991–1018.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[6] R. L. Graham, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.

[7] K. S. Hong and J. Y.-T. Leung, *Some results on Liu's conjecture*, SIAM J. Discrete Math., 5 (1992), pp. 500–523.

[8] C. L. Liu, *Optimal scheduling on multi-processor computing systems*, in Proceedings of the 13th Annual Symposium on Switching and Automata Theory, IEEE Computer Society, Los Alamitos, CA, 1972, pp. 155–160.

[9] R. McNaughton, *Scheduling with deadlines and loss functions*, Management Sci., 6 (1959), pp. 1–12.

[10] J. D. Ullman, *Complexity of sequencing problems*, in Computer and Job-Shop Scheduling Theory, E. G. Coffman, Jr., ed., John Wiley, New York, 1976, pp. 139–164.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 681–699

NONDETERMINISTIC QUANTUM QUERY AND COMMUNICATION COMPLEXITIES*

RONALD DE WOLF†

Abstract. We study nondeterministic quantum algorithms for Boolean functions f. Such algorithms have positive acceptance probability on input x iff $f(x) = 1$. In the setting of query complexity, we show that the nondeterministic quantum complexity of a Boolean function is equal to its "nondeterministic polynomial" degree. We also prove a quantum-vs.-classical gap of 1 vs. n for nondeterministic query complexity for a total function. In the setting of communication complexity, we show that the nondeterministic quantum complexity of a two-party function is equal to the logarithm of the rank of a nondeterministic version of the communication matrix. This implies that the quantum communication complexities of the equality and disjointness functions are $n + 1$ if we do not allow any error probability. We also exhibit a total function in which the nondeterministic quantum communication complexity is exponentially smaller than its classical counterpart.

Key words. quantum computing, query complexity, communication complexity, nondeterminism

AMS subject classification. 68Q10

PII. S0097539702407345

1. Introduction.

1.1. Motivation. In classical computing, *nondeterministic* computation has a prominent place in many different models and for many good reasons. For example, in Turing machine complexity, the study of nondeterminism leads naturally to the class of NP-complete problems, which contains some of the most important and practically relevant computer science problems—as well as some of the hardest theoretical open questions. In fields like query complexity and communication complexity, there is a tight relation between deterministic complexity and nondeterministic complexity, but it is often much easier to analyze upper and lower bounds for the latter than for the former.

Suppose we want to compute a Boolean function f in some algorithmic setting, such as that of Turing machines, decision trees, or communication protocols. Consider the following two ways of viewing a nondeterministic algorithm. The first and most common way is to think of it as a "certificate verifier": a deterministic algorithm A that receives, apart from the input x, a "certificate" y whose validity it needs to verify. For all inputs x, if $f(x) = 1$, then there is a certificate y such that $A(x, y) = 1$; if $f(x) = 0$, then $A(x, y) = 0$ for all y. Second, we may view A as a *randomized* algorithm whose acceptance probability is positive if $f(x) = 1$ and whose acceptance probability is zero if $f(x) = 0$. It is easy to see that these two views are equivalent in the classical case. To turn an algorithm A of the first kind into one of the second kind, we can just guess a certificate y at random and output $A(x, y)$. This will have positive acceptance probability iff $f(x) = 1$. For the other direction, we can consider

*Received by the editors May 8, 2002; accepted for publication (in revised form) December 12, 2002; published electronically April 17, 2003. This paper combines results from the conference papers [50, 29] with some new results.

http://www.siam.org/journals/sicomp/32-3/40734.html

†CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands (rdewolf@cwi.nl). This author was partially supported by the EU fifth framework project QAIP, IST–1999–11234. Part of this paper was written when the author was a postdoc at UC Berkeley, supported by Talent grant S 62–565 from the Netherlands Organization for Scientific Research (NWO).

the sequence of coin flips used by an algorithm of the second kind as a certificate. Clearly, there will be a certificate leading to output 1 iff $f(x) = 1$, which gives us an algorithm of the first kind.

Both views may be generalized to the quantum case, yielding *three* potential definitions of nondeterministic quantum algorithms, possibly nonequivalent. The quantum algorithm may be required to output the right answer $f(x)$ when given an appropriate certificate, which we can take to be either quantum or classical. Or, third, the quantum algorithm may be required to have positive acceptance probability iff $f(x) = 1$. An example is given by two alternative definitions of quantum nondeterminism in the case of quantum Turing machine complexity. Kitaev defines the class "bounded-error quantum-NP" (BNQP) as the set of languages accepted by polynomial-time bounded-error quantum algorithms that are given a polynomial-size quantum certificate (e.g., [32, 31] and [30, Chapter 14]). On the other hand, Adleman, Demarrais, and Huang [2] and Fenner et al. [24] define quantum-NP as the set of languages L for which there is a polynomial-time quantum algorithm whose acceptance probability is positive iff $x \in L$. This quantum class was shown to be equal to the classical counting class co-$\mathrm{C_{=}P}$ [24, 52] using tools from Fortnow and Rogers [25].

In this paper, we adopt the latter view: a nondeterministic quantum algorithm for f is defined to be a quantum algorithm that outputs 1 with positive probability if $f(x) = 1$ and that always outputs 0 if $f(x) = 0$. This definition contrasts with the more traditional view of classical determinism as "certificate verification." The motivation for our choice of definition of quantum nondeterminism is twofold. First, in the appendix, we show that this definition is strictly more powerful than the other two possible definitions in the sense of being able to simulate the other definitions efficiently, while the reverse is not true. Second, it turns out that this definition lends itself to very crisp results. Rather than in the quantum Turing machine setting of Kitaev, Adleman, etc., we study the complexity of nondeterministic algorithms in the query complexity and communication complexity settings. Our main results are exact characterizations of these nondeterministic quantum complexities in algebraic terms and large gaps between quantum and classical complexities in both settings. Our algebraic characterizations can be extended to nontotal functions in the obvious way, but we will stick to total functions in our presentation.

1.2. Query complexity. We first consider the model of query complexity, also known as decision tree complexity or black box complexity. Here the goal is to compute some function $f : \{0,1\}^n \to \{0,1\}$, making as few queries to input bits as possible. Most existing quantum algorithms can naturally be expressed in this model and achieve provable speed-ups over the best classical algorithms. Examples can be found, e.g., in [22, 48, 26, 12, 13, 14] and also include the order-finding problem on which Shor's celebrated factoring algorithm is based [47].

Let $D(f)$ and $Q_E(f)$ denote the query complexities of optimal deterministic and quantum algorithms that compute f exactly. Let $deg(f)$ denote the minimal degree among all multilinear polynomials that represent f. (A polynomial p represents f if $f(x) = p(x)$ for all $x \in \{0,1\}^n$.) The following relations are known. The first inequality is due to Beals et al. [6], the second inequality is obvious, and the last is due to Nisan and Smolensky—unpublished, but described in the survey paper [20].

$$\frac{deg(f)}{2} \leq Q_E(f) \leq D(f) \leq O(deg(f)^4).$$

Thus $deg(f)$, $Q_E(f)$, and $D(f)$ are polynomially related for all total f. (The situation

is very different for partial f [22, 48, 47, 7].) Nisan and Szegedy [42] exhibit a function with a large gap between $D(f) = n$ and $deg(f) = n^{0.6\cdots}$, but no function is known where $Q_E(f)$ is significantly larger than $deg(f)$, and it may in fact be true that $Q_E(f)$ and $deg(f)$ are linearly related. In section 2, we show that the *nondeterministic* versions of $Q_E(f)$ and $deg(f)$ are in fact *equal*:

$$NQ(f) = ndeg(f).$$

Here $NQ(f)$ denotes the query complexity of an optimal nondeterministic quantum algorithm for f, which has nonzero acceptance probability iff $f(x) = 1$. The nondeterministic degree $ndeg(f)$ is the minimal degree of a so-called *nondeterministic* polynomial for f, which is required to be nonzero iff $f(x) = 1$. A note on terminology: the name "nondeterministic polynomial" is based only on analogy with the acceptance probability of a nondeterministic algorithm. This name is less than ideal, since such polynomials have little to do with the traditional view of nondeterminism as certificate verification. Nevertheless, we use this name because any alternatives that we could think of were worse (too verbose or confusing).

Apart from the algebraic characterization of the nondeterministic quantum query complexity $NQ(f)$, we also show that $NQ(f)$ may be much smaller than its classical analogue $N(f)$: we exhibit an f where $NQ(f) = 1$ and $N(f) = n$, which is the biggest possible gap allowed by this model. Accordingly, while the case of exact (or, for that matter, bounded-error) computation allows at most polynomial quantum-classical query complexity gaps for total functions, the nondeterministic case allows *unbounded* gaps.

1.3. Communication complexity. In the case of communication complexity, the goal is for two distributed parties, Alice and Bob, to compute some function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$. Alice receives an $x \in \{0,1\}^n$, and Bob receives a $y \in \{0,1\}^n$, and they want to compute $f(x,y)$, exchanging as few bits of communication as possible. This model was introduced by Yao [53] and is fairly well understood for the case in which Alice and Bob are classical players exchanging classical bits [36]. Much less is known about *quantum* communication complexity, where Alice and Bob have a quantum computer and can exchange qubits. This was first studied by Yao [54], and it was shown later that quantum communication complexity can be significantly smaller than classical communication complexity [21, 17, 5, 44, 16].

Let $Dcc(f)$ and $Qcc_E(f)$ denote the communication required for optimal deterministic classical and exact quantum protocols for computing f, respectively.[1] Here we assume Alice and Bob do not share any randomness or prior entanglement. Let $rank(f)$ be the rank of the $2^n \times 2^n$ communication matrix M_f, which is defined by $M_f(x,y) = f(x,y)$. The following relations are known:

$$\frac{\log rank(f)}{2} \leq Qcc_E(f) \leq Dcc(f).$$

The first inequality follows from work of Kremer [35] and Yao [54], as first noted by Buhrman, Cleve, and Wigderson [17]. (In [19] it is shown that this lower bound also holds if the quantum protocol can make use of unlimited prior entanglement between Alice and Bob.) It is an open question whether $Dcc(f)$ can in turn be upper bounded

[1] The notation $D(f)$ is used for deterministic complexity in decision tree complexity as well as in communication complexity. To avoid confusion, we will consistently add "cc" to indicate communication complexity.

by some polynomial in $\log rank(f)$. The conjecture that it can is known as the *log-rank conjecture*. If this conjecture holds, then $Dcc(f)$ and $Qcc_E(f)$ are polynomially related for all total f (which may well be true). It is known that $\log rank(f)$ and $Dcc(f)$ are not linearly related [43]. In section 3, we show that the *nondeterministic* version of $\log rank(f)$ in fact fully determines the nondeterministic version of $Qcc_E(f)$:

$$NQcc(f) = \lceil \log nrank(f) \rceil + 1.$$

Here $nrank(f)$ denotes the minimal rank of a matrix whose (x, y)-entry is nonzero iff $f(x, y) = 1$. Thus we can characterize the nondeterministic quantum communication complexity fully by the logarithm of the rank of its nondeterministic matrix. As far as we know, only two other log-rank-style characterizations of certain variants of communication complexity are known: the communication complexity of quantum sampling due to Ambainis et al. [5] and the so-called modular communication complexity due to Meinel and Waack [38].

Equality and disjointness both have nondeterministic rank 2^n, so their nondeterministic complexities are maximal: $NQcc(\text{EQ}) = NQcc(\text{DISJ}) = n+1$. Since $NQcc(f)$ lower bounds $Qcc_E(f)$, we also obtain optimal bounds for the exact quantum communication complexity of equality and disjointness. In particular, for the equality function, we get $Qcc_E(\text{EQ}) = n + 1$, which answers a question posed by Gilles Brassard in a personal communication [10]. Surprisingly, no proof of this fact seems to be known that avoids our detour via nondeterministic computation. Thus our methods also give new lower bounds for regular quantum communication complexity.

Finally, analogous to the query complexity case, we also show an exponential gap between quantum and classical nondeterministic communication complexity: we exhibit an f where $NQcc(f) \leq \log(n + 1) + 1$ and $Ncc(f) \in \Omega(n)$. Massar et al. [37] earlier found another gap that is unbounded, yet in some sense smaller: $NQcc(\text{NE}) = 2$ versus $Ncc(\text{NE}) = \log n + 1$, where NE is the nonequality function.

2. Nondeterministic quantum query complexity.

2.1. Functions and polynomials. For $x \in \{0, 1\}^n$, we use $|x|$ for the Hamming weight (number of 1's) of x, and x_i for its ith bit, $i \in [n] = \{1, \ldots, n\}$. We use $\vec{0}$ for a string of n zeros. If $B \subseteq [n]$ is a set of (indices of) variables, then x^B denotes the input obtained from x by complementing all variables in B. If $x, y \in \{0, 1\}^n$, then $x \wedge y$ denotes the n-bit string obtained by bitwise ANDing x and y. Let $f : \{0, 1\}^n \to \{0, 1\}$ be a total Boolean function. For example, $\text{OR}(x) = 1$ iff $|x| \geq 1$, $\text{AND}(x) = 1$ iff $|x| = n$, $\text{PARITY}(x) = 1$ iff $|x|$ is odd. We use $\overline{f}$ for the function $1 - f$.

For $b \in \{0, 1\}$, a *b-certificate* for f is an assignment $C : S \to \{0, 1\}$ to some set S of variables, such that $f(x) = b$ whenever x is consistent with C. The *size* of C is $|S|$. The *certificate complexity* $C_x(f)$ of f on input x is the minimal size of an $f(x)$-certificate that is consistent with x. We define the 1-certificate complexity of f as $C^{(1)}(f) = \max_{x:f(x)=1} C_x(f)$. We define $C^{(0)}(f)$ similarly. For example, $C^{(1)}(\text{OR}) = 1$ and $C^{(0)}(\text{OR}) = n$, but $C^{(1)}(\overline{\text{OR}}) = n$ and $C^{(0)}(\overline{\text{OR}}) = 1$.

An n-variate *multilinear polynomial* is a function $p : \mathbb{C}^n \to \mathbb{C}$ that can be written

$$p(x) = \sum_{S \subseteq [n]} a_S X_S.$$

Here S ranges over all sets of indices of variables, a_S is a complex number, and the monomial X_S is the product $\Pi_{i \in S} x_i$ of all variables in S. The *degree* $deg(p)$

of p is the degree of a largest monomial with nonzero coefficient. It is well known that every total Boolean f has a unique polynomial p such that $p(x) = f(x)$ for all $x \in \{0,1\}^n$. Let $deg(f)$ be the degree of this polynomial, which is at most n. For example, $\mathrm{OR}(x_1, x_2) = x_1 + x_2 - x_1x_2$, which has degree 2. Every multilinear polynomial $p = \sum_S a_S X_S$ can also be written out uniquely in the so-called *Fourier basis*:

$$p(x) = \sum_S c_S(-1)^{x \cdot S}.$$

Again S ranges over all sets of indices of variables (we often identify a set S with its characteristic n-bit vector), c_S is a complex number, and $x \cdot S$ denotes the inner product of the n-bit strings x and S, or, equivalently, $x \cdot S = |x \wedge S| = \sum_{i \in S} x_i$. It is easy to see that $deg(p) = \max\{|S| \mid c_S \neq 0\}$. For example, $\mathrm{OR}(x_1, x_2) = \frac{3}{4} - \frac{1}{4}(-1)^{x_1} - \frac{1}{4}(-1)^{x_2} - \frac{1}{4}(-1)^{x_1+x_2}$ in the Fourier basis. We refer to [8, 42, 20] for more details about polynomial representations of Boolean functions.

We introduce the notion of a *nondeterministic polynomial* for f. This is a polynomial p such that $p(x) \neq 0$ iff $f(x) = 1$. Let the *nondeterministic degree* of f, denoted $ndeg(f)$, be the minimum degree among all nondeterministic polynomials p for f. For example, $p(x) = \sum_{i=1}^n x_i$ is a nondeterministic polynomial for OR; hence $ndeg(\mathrm{OR}) = 1$.

We mention some upper and lower bounds for $ndeg(f)$. Let f be a nonconstant symmetric function (i.e., $f(x)$ depends only on $|x|$). Suppose f achieves value 0 on the z Hamming weights, $k_1, \ldots, k_z$. Since $|x| = \sum_i x_i$, it is easy to see that $(|x|-k_1)(|x|-k_2)\cdots(|x|-k_z)$ is a nondeterministic polynomial for f; hence $ndeg(f) \leq z$. This upper bound is tight for AND (see below) but not for PARITY. For example, $p(x_1, x_2) = x_1 - x_2$ is a degree-1 nondeterministic polynomial for PARITY on two variables: it assumes value 0 on x-weights 0 and 2 and ± 1 on weight 1. By squaring $p(x)$ and then using standard symmetrization techniques (as used, for instance, in [39, 42, 6]), we can also show the general lower bound $ndeg(f) \geq z/2$ for symmetric f. Furthermore, it is easy to show that $ndeg(f) \leq C^{(1)}(f)$ for every f. (Take a polynomial that is the "sum" over all 1-certificates for f.)

Finally, we mention a general lower bound on $ndeg(f)$. Let $\Pr[p \neq 0] = |\{x \in \{0,1\}^n \mid p(x) \neq 0\}|/2^n$ denote the probability that a random Boolean input x makes a function p nonzero. A lemma of Schwartz [46] (see also [42, section 2.2]) states that if p is a nonconstant multilinear polynomial of degree d, then $\Pr[p \neq 0] \geq 2^{-d}$, and hence $d \geq \log(1/\Pr[p \neq 0])$. Since a nondeterministic polynomial p for f is nonzero iff $f(x) = 1$, it follows that

$$ndeg(f) \geq \log(1/\Pr[f \neq 0]) = \log(1/\Pr[f = 1]).$$

Accordingly, functions with a very small fraction of 1-inputs will have high nondeterministic degree. For instance, $\Pr[\mathrm{AND} = 1] = 2^{-n}$, so $ndeg(\mathrm{AND}) = n$.

2.2. Quantum computing. We assume familiarity with classical computation and briefly sketch the setting of quantum computation (see, e.g., [40] for more details). An *m-qubit state* is a linear combination of all classical m-bit states

$$|\phi\rangle = \sum_{i \in \{0,1\}^m} \alpha_i |i\rangle,$$

where $|i\rangle$ denotes the basis state i (a classical m-bit string) and α_i is a complex number that is called the *amplitude* of $|i\rangle$. We require $\sum_i |\alpha_i|^2 = 1$. Viewing $|\phi\rangle$ as

a 2^m-dimensional column vector, we use $\langle\phi|$ for the row vector that is the conjugate transpose of $|\phi\rangle$. Note that the inner product $\langle i||j\rangle = \langle i|j\rangle$ is 1 if $i = j$ and 0 if $i \neq j$. When we observe $|\phi\rangle$, we will see $|i\rangle$ with probability $|\langle i|\phi\rangle|^2 = |\alpha_i|^2$, and the state will collapse to the observed $|i\rangle$. A quantum operation which is not an observation corresponds to a unitary (i.e., norm-preserving) transformation U on the 2^m-dimensional vector of amplitudes.

2.3. Query complexity. Suppose we want to compute some function $f : \{0,1\}^n \to \{0,1\}$. For input $x \in \{0,1\}^n$, a *query* corresponds to the unitary transformation O that maps $|i,b,z\rangle \to |i,b \oplus x_i,z\rangle$. Here $i \in [n]$ and $b \in \{0,1\}$; the z-part corresponds to the workspace, which is not affected by the query. We assume that the input can be accessed only via such queries. A T-query quantum algorithm has the form $A = U_T O U_{T-1} \cdots O U_1 O U_0$, where the U_k are fixed unitary transformations, independent of the input x. This A depends on x via the T applications of O. We sometimes write A_x to emphasize this. The algorithm starts in initial state $|\vec{0}\rangle$, and its *output* is the bit obtained from observing the leftmost qubit of the final superposition $A|\vec{0}\rangle$. The *acceptance probability* of A (on input x) is its probability of outputting 1 (on x).

We will consider classical and quantum algorithms and will count only the number of queries these algorithms make on a worst-case input. Let $D(f)$ and $Q_E(f)$ be the query complexities of optimal deterministic classical and exact quantum algorithms for computing f, respectively. $D(f)$ is also known as the decision tree complexity of f. Similarly we can define $R_2(f)$ and $Q_2(f)$ to be the query complexity of f for bounded-error classical and quantum algorithms, respectively. Quantum query complexity and its relation to classical complexity has been well studied in recent years; see, for example, [6, 4, 20].

We define a *nondeterministic algorithm* for f to be an algorithm that has positive acceptance probability on input x iff $f(x) = 1$. Let $N(f)$ and $NQ(f)$ be the query complexities of optimal nondeterministic classical and quantum algorithms for f, respectively. It is easy to show that the 1-certificate complexity fully characterizes the classical nondeterministic complexity of f.

PROPOSITION 2.1. $N(f) = C^{(1)}(f)$.

Proof. A classical algorithm that guesses a 1-certificate, queries its variables, and outputs 1 iff the certificate holds is a nondeterministic algorithm for f. Hence $N(f) \leq C^{(1)}(f)$.

A nondeterministic algorithm for f can only output 1 if the outcomes of the queries that it has made force the function to 1. Hence, if x is an input where all 1-certificates have size at least $C^{(1)}(f)$, then the algorithm will have to query at least $C^{(1)}(f)$ variables before it can output 1 (which it must do on some runs). Hence $N(f) \geq C^{(1)}(f)$. □

2.4. Algebraic characterization. Here we show that $NQ(f)$ is equal to $ndeg(f)$, using the following result from [6].

LEMMA 2.2 (see [6]). *The amplitudes of the basis states in the final superposition of a T-query quantum algorithm can be written as multilinear complex-valued polynomials of degree $\leq T$ in the n x_i-variables. Therefore, the acceptance probability of the algorithm (which is the sum of squares of some of those amplitudes) can be written as an n-variate multilinear polynomial $P(x)$ of degree $\leq 2T$.*

Note that the acceptance probability of a nondeterministic quantum algorithm is actually a nondeterministic polynomial for f, since it is positive iff $f(x) = 1$. By Lemma 2.2, this polynomial will have degree at most twice the number of queries

of the algorithm, which immediately implies $ndeg(f)/2 \leq NQ(f)$. Below we will show how we can get rid of the factor $1/2$ in this lower bound, improving it to $ndeg(f) \leq NQcc(f)$. We show that this lower bound is in fact optimal by deriving a nondeterministic algorithm from a nondeterministic polynomial. This derivation uses a trick similar to the one used in [24] to show that co-C$_=$P $\subseteq$ quantum-NP.

THEOREM 2.3. *$NQ(f) = ndeg(f)$.*

Proof. Upper bound. Let $p(x)$ be a nondeterministic polynomial for f of degree $d = ndeg(f)$. Recall that $x \cdot S$ denotes $|x \wedge S|$, identifying $S \subseteq [n]$ with its characteristic n-bit vector. We write p in the Fourier basis:

$$p(x) = \sum_S c_S (-1)^{x \cdot S}.$$

Since $deg(p) = \max\{|S| \mid c_S \neq 0\}$, we have that $c_S \neq 0$ only if $|S| \leq d$.

We can construct a unitary transformation F that uses d queries to x and maps $|S\rangle \rightarrow (-1)^{x \cdot S}|S\rangle$ whenever $|S| \leq d$. Informally, this transformation does a controlled parity-computation: it computes $|x \cdot S| \pmod 2$ using $|S|/2$ queries [6, 23], then adds a phase "-1" if that answer is 1, and then reverses the computation to clean up the workspace and the answer at the cost of another $|S|/2$ queries. (If $|S|$ is odd, then one variable is treated separately, still using $|S|$ queries in total.)

Now consider the following quantum algorithm:

1. Start with $c \sum_S c_S |S\rangle$ (an n-qubit state, where $c = 1/\sqrt{\sum_S |c_S|^2}$ is a normalizing constant).
2. Apply F to the state.
3. Apply a Hadamard transform H to each qubit.
4. Measure the final state, and output 1 if the outcome is the all-zero state $|\vec{0}\rangle$, and output 0 otherwise.

The state after step 2 is $c \sum_S c_S (-1)^{x \cdot S} |S\rangle$. Note that the sum of the amplitudes in this state is $c \cdot p(x)$, which is nonzero iff $f(x) = 1$. The Hadamard transform in step 3 gives us this sum as amplitude of the $|\vec{0}\rangle$-state, with a normalizing factor of $1/\sqrt{2^n}$. Accordingly, the probability of observing $|\vec{0}\rangle$ at the end is

$$\begin{aligned} P(x) &= \left| \langle \vec{0} | H^{\otimes n} F c \sum_S c_S |S\rangle \right|^2 \\ &= \frac{c^2}{2^n} \left| \sum_{S'} \langle S' | \sum_S c_S (-1)^{x \cdot S} |S\rangle \right|^2 \\ &= \frac{c^2}{2^n} \left| \sum_S c_S (-1)^{x \cdot S} \right|^2 \\ &= \frac{c^2 p(x)^2}{2^n}. \end{aligned}$$

Since $p(x)$ is nonzero iff $f(x) = 1$, $P(x)$ will be positive iff $f(x) = 1$. Hence we have a nondeterministic quantum algorithm for f with $d = ndeg(f)$ queries.

Lower bound. Let $T = NQ(f)$, and consider a T-query nondeterministic quantum algorithm for f. By Lemma 2.2, the amplitudes α_i in the final state,

$$|\phi^x\rangle = \sum_i \alpha_i(x) |i\rangle,$$

on input x are n-variate polynomials of x of degree $\leq T$. We use the probabilistic method [3] to show that some linear combination of these polynomials is a nondeterministic polynomial for f, thus avoiding losing the factor $1/2$ mentioned after Lemma 2.2.

Let S be the set of basis states having a 1 as leftmost bit (observing such a state will lead the algorithm to output 1). Since the algorithm is nondeterministic, we have the following properties:

If $f(x) = 0$, then $\alpha_i(x) = 0$ for all $i \in S$.

If $f(x) = 1$, then $\alpha_i(x) \neq 0$ for at least one $i \in S$.

Let I be an arbitrary set of more than 2^n numbers. For each $i \in S$, pick a coefficient c_i uniformly at random from I, and define $p(x) = \sum_{i \in S} c_i \alpha_i(x)$. By the first property, we have $p(x) = 0$ whenever $f(x) = 0$. Now consider an x for which $f(x) = 1$, and let $k \in S$ satisfy $a = \alpha_k(x) \neq 0$. Such a k must exist by the second property. We want to show that the event $p(x) = 0$ happens only with very small probability (probability taken over the random choices of the c_i). In order to do this, we fix the random choices c_i for all $i \neq k$ and view $p(x) = ac_k + b$ as a linear function in the only not-yet-chosen coefficient c_k. Since $a \neq 0$, at most one out of $|I| > 2^n$ many possible choices of c_k can make $p(x) = 0$, so

$$\Pr[p(x) = 0] < 2^{-n}.$$

However, then, by the union bound we have

$$\Pr\left[\text{there is an } x \in f^{-1}(1) \text{ for which } p(x) = 0\right] \leq \sum_{x \in f^{-1}(1)} \Pr[p(x) = 0] < 2^n \cdot 2^{-n} = 1.$$

This probability is strictly less than 1, which shows that there exists a way of setting the coefficients c_i that satisfies $p(x) \neq 0$ for all $x \in f^{-1}(1)$, thus making p a nondeterministic polynomial for f. Since p is a sum of polynomials of degree $\leq T$, it follows that $ndeg(f) \leq deg(p) \leq T = NQ(f)$. □

2.5. Quantum-classical separation. What is the biggest possible gap between quantum and classical nondeterministic query complexity? Consider the total Boolean function $f : \{0,1\}^n \to \{0,1\}$ defined by

$$f(x) = 1 \text{ iff } |x| \neq 1.$$

It is easy to see that $N(f) = C^{(1)}(f) = C^{(0)}(f) = n$. On the other hand, the following is a degree-1 nondeterministic polynomial for f:

$$p(x) = \left(\sum_{i=1}^{n} x_i\right) - 1 = \frac{n}{2} - 1 - \frac{1}{2}\sum_{i=1}^{n}(-1)^{x_i}. \tag{2.1}$$

Thus we have that $NQ(f) = ndeg(f) = 1$. Explicitly, the 1-query algorithm that we get from the proof is as follows:

1. Start with $c\,((n/2-1)|\vec{0}\rangle - (1/2)\sum_i |e_i\rangle)$, where $c = 1/\sqrt{n^2/4 - 3n/4 + 1}$ and $|e_i\rangle$ has a 1 only at the ith bit.
2. Using one query, we can map $|e_i\rangle \to (-1)^{x_i}|e_i\rangle$.
3. Applying a Hadamard transform turns the amplitude of $|\vec{0}\rangle$ into $\alpha_{\vec{0}} = \frac{c}{\sqrt{2^n}}\,((n/2-1) - \sum_i (-1)^{x_i}/2) = cp(x)/\sqrt{2^n}$.

4. Hence the probability of observing $|\vec{0}\rangle$ at the end is $\alpha_{\vec{0}}^2 = c^2 p(x)^2/2^n$.

For the complement of f, we can easily show $NQ(\overline{f}) = ndeg(\overline{f}) \geq n-1$ (the "-1" is tight for $n=2$; witness $p(x) = x_1 - x_2$). In sum, we have the following theorem.

THEOREM 2.4. *For the above f, we have $NQ(f) = 1$, $NQ(\overline{f}) \geq n-1$, and $N(f) = N(\overline{f}) = n$.*

2.6. Relation to some other complexity measures. Many relations are known between all sorts of complexity measures of Boolean functions, such as polynomial degree, certificate complexity, various classical and quantum decision tree complexities, etc. A survey may be found in [20]. In this subsection, we will similarly embed $ndeg(f)$ $(= NQ(f))$ in this web of relations and give upper bounds on $D(f)$ in terms of $ndeg(f)$, $C(f)$, and the *block sensitivity* $bs(f)$, which is defined as follows. A set of (indices of) variables $B \subseteq [n]$ is called a *sensitive block* for f on input x if $f(x) \neq f(x^B)$; B is *minimal* if no $B' \subset B$ is sensitive. The block sensitivity $bs_x(f)$ is the maximal number of disjoint minimal sensitive blocks in x, and $bs^{(b)}(f) = \max_{x \in f^{-1}(b)} bs_x(f)$.

LEMMA 2.5. *If $f(x) = 0$ and B is a minimal sensitive block for f on x, then $|B| \leq ndeg(f)$.*

Proof. Assume without loss of generality that $x = \vec{0}$. Because B is minimal, for every proper subset B' of B, we have $f(x) = f(x^{B'}) = 0$, but on the other hand $f(x^B) = 1$. Accordingly, if we fix all variables outside of B to zero, then we obtain the AND-function of $|B|$ variables, which requires nondeterministic degree $|B|$. Hence $|B| \leq ndeg(f)$. □

LEMMA 2.6. $C^{(0)}(f) \leq bs^{(0)}(f) ndeg(f)$.

Proof. Consider any input x. As Nisan [41] proved, the union of a maximal set of sensitive blocks forms a certificate for that input (for otherwise there would be one more sensitive block). If $f(x) = 0$, then there can be at most $bs^{(0)}(f)$ disjoint sensitive blocks, and by the previous lemma each block contains at most $ndeg(f)$ variables. Hence each 0-input contains a certificate of at most $bs^{(0)}(f) ndeg(f)$ variables. □

The following theorem improves upon an argument of Nisan and Smolensky, described in [20].

THEOREM 2.7. $D(f) \leq C^{(0)}(f) ndeg(f)$.

Proof. Let p be a nondeterministic polynomial for f of degree $d = ndeg(f)$. Note that if we take a 0-certificate $C : S \to \{0,1\}$ and fix the S-variables accordingly, then p must reduce to the constant-0 polynomial. This implies that S intersects all degree-d monomials of p, because a nonintersected degree-d monomial would still be present in the reduced polynomial, which would then not be constant-0. Thus taking a minimal 0-certificate and querying its variables reduces the degree of p by at least 1. Repeating this at most $ndeg(f)$ times, we reduce p to a constant polynomial and know $f(x)$. This algorithm takes at most $C^{(0)}(f) ndeg(f)$ queries. □

Combining this with the fact that $bs^{(0)}(f) \leq 6Q_2(f)^2$ [6], we obtain the following.

COROLLARY 2.8. $D(f) \leq bs^{(0)}(f) ndeg(f)^2 \leq 6\, Q_2(f)^2 NQ(f)^2$.

This corollary has the somewhat paradoxical consequence that if the nondeterministic complexity $NQ(f)$ is small, then the bounded-error complexity $Q_2(f)$ must be large (i.e., close to $D(f)$). For instance, if $NQ(f) = O(1)$, then $Q_2(f) = \Omega(\sqrt{D(f)})$. We hope that this result will help tighten the relation $D(f) = O(Q_2(f)^6)$ that was proved in [6].

3. Nondeterministic quantum communication complexity.

3.1. Communication complexity. In the standard version of communication complexity, two parties (Alice and Bob) want to compute some function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$. For example, $\mathrm{EQ}(x,y) = 1$ iff $x = y$, $\mathrm{NE}(x,y) = 1$ iff $x \neq y$, and $\mathrm{DISJ}(x,y) = 1$ iff $|x \wedge y| = 0$. A *rectangle* is a subset $R = S \times T$ of the domain of f. R is a 1-*rectangle* (for f) if $f(x,y) = 1$ for all $(x,y) \in R$. A 1-*cover* for f is a set of 1-rectangles whose union contains all 1-inputs of f. $Cov^1(f)$ denotes the minimal size (i.e., minimal number of rectangles) of a 1-cover for f. Similarly, we define 0-rectangles, 0-covers, and $Cov^0(f)$.

The *communication matrix* M_f of f is the $2^n \times 2^n$ Boolean matrix whose (x,y)-entry is $f(x,y)$, and *rank*(f) denotes the rank of M_f over the field of complex numbers. A $2^n \times 2^n$ matrix M is called a *nondeterministic communication matrix* for f if it has the property that $M(x,y) \neq 0$ iff $f(x,y) = 1$. Thus M is any matrix obtainable by replacing 1-entries in M_f by nonzero complex numbers. Let the *nondeterministic rank* of f, denoted *nrank*(f), be the minimum rank (over the complex field) among all nondeterministic matrices M for f.[2]

We consider classical and quantum communication protocols and count only the amount of communication (bits or qubits) that these protocols make on a worst-case input. For classical communication protocols, we refer to [36]. Here we briefly define quantum communication protocols, referring to the surveys [49, 15, 33, 11, 51] for more details. The space in which the quantum protocol works consists of three parts: Alice's part, the communication channel, and Bob's part. (We do not write the dimensions of these spaces explicitly.) Initially these three parts contain only 0-qubits,

$$|0\rangle|0\rangle|0\rangle.$$

We assume Alice starts the protocol. She applies a unitary transformation $U_1^A(x)$ to her private space and part of the channel. This corresponds to her initial computation and her first message. The length of this message is the number of channel qubits on which $U_1^A(x)$ acts. The total state is now

$$(U_1^A(x) \otimes I^B)|0\rangle|0\rangle|0\rangle,$$

where $\otimes$ denotes tensor product, and I^B denotes the identity transformation on Bob's part. Then Bob applies a unitary transformation $U_2^B(y) = V_2^B(y)S_2^B$ to his part and the channel. First, the operation S_2^B "reads" Alice's message by swapping the contents of the channel with some fresh $|0\rangle$-qubits in Bob's private space. After this, the unitary $V_2^B(y)$ is applied to Bob's private space and part of the channel. This corresponds to Bob's private computation and his putting a message to Alice on the channel. The length of this new message is the number of channel-qubits on which

[2] This definition looks somewhat similar to the definition of the *Colin de Verdiére parameter* $\mu(G)$ of an undirected graph G [27]. For $G = (V,E)$ with $|V| = n$, $\mu(G)$ is defined to be the maximal corank ($= n - rank$) among all real symmetric $n \times n$ matrices M having the following three properties: (1) $M_{ij} < 0$ if $(i,j) \in E$ and $M_{ij} = 0$ if $i \neq j$ and $(i,j) \notin E$; (2) M has exactly one negative eigenvalue of multiplicity 1; (3) there is no real symmetric matrix $X \neq 0$ such that $MX = 0$ and $X_{ij} = 0$ whenever $i = j$ or $M_{ij} \neq 0$. Such a matrix M is a nondeterministic matrix for the communication complexity problem $f : [n] \times [n] \to \{0,1\}$ defined by $f(i,j) = 1$ iff $(i,j) \in E$, with the promise that the inputs i and j are distinct. However, the Colin de Verdière requirement appears to be more stringent, since it constrains the nondeterministic matrix further by properties (2) and (3).

$V_2^B(y)$ acts. This process goes back and forth for some k messages, so the final state of the protocol on input (x,y) will be (in case Alice goes last as well)

$$(U_k^A(x)\otimes I^B)(I^A\otimes U_{k-1}^B(y))\cdots(I^A\otimes U_2^B(y))(U_1^A(x)\otimes I^B)|0\rangle|0\rangle|0\rangle.$$

The total *cost* of the protocol is the total length of all messages sent, on a worst-case input (x,y). For technical convenience, we assume that at the end of the protocol the output bit is the first qubit on the channel. Thus the acceptance probability $P(x,y)$ of the protocol is the probability that a measurement of the final state gives a "1" in the first channel-qubit. Note that we do not allow intermediate measurements during the protocol. This is without loss of generality; it is well known that such measurements can be postponed until the end of the protocol at no extra communication cost.

Let $Dcc(f)$ and $Qcc_E(f)$ be the communication complexities of optimal deterministic classical and quantum protocols for computing f, respectively. A *nondeterministic protocol* for f is a protocol that has positive acceptance probability on input (x,y) iff $f(x,y)=1$. Let $Ncc(f)$ and $NQcc(f)$ be the communication complexities of optimal nondeterministic classical and quantum protocols for f, respectively. Our $Ncc(f)$ is called $N^1(f)$ in [36].

It is not hard to show that $Ncc(f)=\lceil\log Cov^1(f)\rceil+1$, where the "+1" is due to the fact that we want Alice and Bob both to know the output at the end of the protocol.

3.2. Algebraic characterization. Here we characterize $NQcc(f)$ in terms of $nrank(f)$. We use the following lemma. It was stated without proof by Yao [54] and in more detail by Kremer [35] and is key to many of the earlier lower bounds on quantum communication complexity as well as to ours. It is easily proven by induction on ℓ.

LEMMA 3.1 (see Yao [54] and Kremer [35]). *The final state of an ℓ-qubit protocol on input (x,y) can be written as*

$$\sum_{i\in\{0,1\}^\ell}|A_i(x)\rangle|i_\ell\rangle|B_i(y)\rangle,$$

where the $A_i(x),B_i(y)$ are vectors (of norm ≤ 1), and i_ℓ denotes the last bit of the ℓ-bit string i (the output bit).

The acceptance probability $P(x,y)$ of the protocol is the squared norm of the part of the final state that has $i_\ell=1$. Letting a_{ij} be the 2^n-dimensional complex column vector with the inner products $\langle A_i(x)|A_j(x)\rangle$ as entries and b_{ij} the 2^n-dimensional column vector with entries $\langle B_i(y)|B_j(y)\rangle$, we can write P (viewed as a $2^n\times 2^n$ matrix) as the sum $\sum_{i,j:i_\ell=j_\ell=1}a_{ij}b_{ij}^T$ of $2^{2\ell-2}$ matrices, each of rank at most 1, so the rank of P is at most $2^{2\ell-2}$. For example, for exact protocols this gives immediately that $\ell\ge\frac{1}{2}\log rank(f)+1$, and for nondeterministic protocols $\ell\ge\frac{1}{2}\log nrank(f)+1$.

Below we show how we can get rid of the factor $\frac{1}{2}$ in the nondeterministic case and show that the lower bound of $\log nrank(f)+1$ is actually optimal. The lower bound part of the proof relies on the following technical lemma.

LEMMA 3.2. *If there exist two families of vectors $\{A_1(x),\ldots,A_m(x)\}\subseteq\mathbb{C}^d$ and $\{B_1(y),\ldots,B_m(y)\}\subseteq\mathbb{C}^d$ such that, for all $x\in\{0,1\}^n$ and $y\in\{0,1\}^n$, we have*

$$\sum_{i=1}^m A_i(x)\otimes B_i(y)=0 \text{ iff } f(x,y)=0,$$

then $nrank(f)\le m$.

Proof. Assume there exist two such families of vectors. Let $A_i(x)_j$ denote the jth entry of vector $A_i(x)$, and similarly let $B_i(y)_k$ denote the kth entry of vector $B_i(y)$. We use pairs $(j,k) \in \{1,\ldots,d\}^2$ to index entries of vectors in the d^2-dimensional tensor space. Note that

if $f(x,y)=0$, then $\sum_{i=1}^m A_i(x)_j B_i(y)_k = 0$ for all (j,k), and
if $f(x,y)=1$, then $\sum_{i=1}^m A_i(x)_j B_i(y)_k \neq 0$ for some (j,k).

As a first step, we want to replace the vectors $A_i(x)$ and $B_i(y)$ by numbers $a_i(x)$ and $b_i(y)$ that have similar properties. We use the probabilistic method to show that this can be done.

Let I be an arbitrary set of 2^{2n+1} numbers. Choose coefficients $\alpha_1,\ldots,\alpha_d$ and $\beta_1,\ldots,\beta_d$, each coefficient picked uniformly at random from I. For every x define $a_i(x) = \sum_{j=1}^d \alpha_j A_i(x)_j$, and for every y define $b_i(y) = \sum_{k=1}^d \beta_k B_i(y)_k$. Consider the number

$$v(x,y) = \sum_{i=1}^m a_i(x)b_i(y) = \sum_{j,k=1}^d \alpha_j\beta_k \left(\sum_{i=1}^m A_i(x)_j B_i(y)_k\right).$$

If $f(x,y)=0$, then $v(x,y)=0$ for all choices of the α_j,β_k.

Now consider some (x,y) with $f(x,y)=1$. There is a pair (j',k') for which $\sum_{i=1}^m A_i(x)_{j'} B_i(y)_{k'} \neq 0$. We want to prove that $v(x,y)=0$ happens only with very small probability. In order to do this, fix the random choices of all α_j, $j \neq j'$, and β_k, $k \neq k'$, and view $v(x,y)$ as a function of the two remaining not-yet-chosen coefficients $\alpha = \alpha_{j'}$ and $\beta = \beta_{k'}$,

$$v(x,y) = c_0\alpha\beta + c_1\alpha + c_2\beta + c_3.$$

Here we know that $c_0 = \sum_{i=1}^m A_i(x)_{j'} B_i(y)_{k'} \neq 0$. There is at most one value of α for which $c_0\alpha + c_2 = 0$. All other values of α turn $v(x,y)$ into a linear equation in β, so for those α there is at most one choice of β that gives $v(x,y)=0$. Hence out of the $(2^{2n+1})^2$ different ways of choosing (α,β), at most $2^{2n+1} + (2^{2n+1}-1)\cdot 1 < 2^{2n+2}$ choices give $v(x,y)=0$. Therefore,

$$\Pr[v(x,y)=0] < \frac{2^{2n+2}}{(2^{2n+1})^2} = 2^{-2n}.$$

Using the union bound, we now have

$$\Pr\left[\text{there is an } (x,y)\in f^{-1}(1) \text{ for which } v(x,y)=0\right] \leq \sum_{(x,y)\in f^{-1}(1)} \Pr[v(x,y)=0] < 2^{2n}\cdot 2^{-2n} = 1.$$

This probability is strictly less than 1, so there exist sets $\{a_1(x),\ldots,a_m(x)\}$ and $\{b_1(y),\ldots,b_m(y)\}$ that make $v(x,y)\neq 0$ for every $(x,y)\in f^{-1}(1)$. We thus have that

$$\sum_{i=1}^m a_i(x)b_i(y) = 0 \text{ iff } f(x,y)=0.$$

View the a_i and b_i as 2^n-dimensional vectors, let A be the $2^n \times m$ matrix having the a_i as columns, and let B be the $m \times 2^n$ matrix having the b_i as rows. Then $(AB)_{xy} = \sum_{i=1}^m a_i(x)b_i(y)$, which is 0 iff $f(x,y)=0$. Thus AB is a nondeterministic matrix for f, and $nrank(f) \leq rank(AB) \leq rank(A) \leq m$. ∎

Lemma 3.2 allows us to prove the following tight characterization.

THEOREM 3.3. $NQcc(f) = \lceil \log nrank(f) \rceil + 1$.

Proof. Upper bound. Let $r = nrank(f)$, and let M be a rank-r nondeterministic matrix for f. Let $M^T = U\Sigma V$ be the singular value decomposition of the transpose of M [28], so U and V are unitary, and Σ is a diagonal matrix whose first r diagonal entries are positive real numbers and whose other diagonal entries are 0. Below we describe a one-round nondeterministic protocol for f, using $\lceil \log r \rceil + 1$ qubits.

First, Alice prepares the state $|\phi_x\rangle = c_x \Sigma V |x\rangle$, where $c_x > 0$ is a normalizing real number that depends on x. Because only the first r diagonal entries of Σ are nonzero, only the first r amplitudes of $|\phi_x\rangle$ are nonzero, so $|\phi_x\rangle$ can be compressed into $\lceil \log r \rceil$ qubits. Alice sends these qubits to Bob. Bob then applies U to $|\phi_x\rangle$ and measures the resulting state. If he observes $|y\rangle$, then he puts 1 on the channel, and otherwise he puts 0 there. The acceptance probability of this protocol is

$$P(x,y) = |\langle y|U|\phi_x\rangle|^2 = c_x^2 |\langle y|U\Sigma V|x\rangle|^2 = c_x^2 |M_{yx}^T|^2 = c_x^2 |M_{xy}|^2.$$

Since M_{xy} is nonzero iff $f(x,y) = 1$, $P(x,y)$ will be positive iff $f(x,y) = 1$. Thus we have a nondeterministic quantum protocol for f with $\lceil \log r \rceil + 1$ qubits of communication.

Lower bound. Consider a nondeterministic ℓ-qubit protocol for f. By Lemma 3.1, its final state on input (x,y) can be written as

$$\sum_{i \in \{0,1\}^\ell} |A_i(x)\rangle |i_\ell\rangle |B_i(y)\rangle.$$

Without loss of generality, we assume the vectors $A_i(x)$ and $B_i(y)$ all have the same dimension d. Let $S = \{i \in \{0,1\}^\ell \mid i_\ell = 1\}$, and consider the part of the state that corresponds to output 1 (we drop the $i_\ell = 1$ and the $|\cdot\rangle$-notation here),

$$\phi(x,y) = \sum_{i \in S} A_i(x) \otimes B_i(y).$$

Because the protocol has acceptance probability 0 iff $f(x,y) = 0$, this vector $\phi(x,y)$ will be the zero vector iff $f(x,y) = 0$. The previous lemma gives $nrank(f) \leq |S| = 2^{\ell-1}$; hence $\log(nrank(f)) + 1 \leq NQcc(f)$. □

Note that any nondeterministic matrix for the equality function has nonzeros on its diagonal and zeros off-diagonal and hence has full rank. Thus we obtain $NQcc(\mathrm{EQ}) = n + 1$. Similarly, a nondeterministic matrix for disjointness has full rank, because reversing the ordering of the columns in M_f gives an upper triangular matrix with nonzero elements on the diagonal. This gives tight bounds for the nondeterministic as well as for the exact setting, neither of which was known prior to this work.

COROLLARY 3.4. $Qcc_E(\mathrm{EQ}) = NQcc(\mathrm{EQ}) = n+1$ *and* $Qcc_E(\mathrm{DISJ}) = NQcc(\mathrm{DISJ}) = n + 1$.

3.3. Quantum-classical separation. To repeat, classically we have $Ncc(f) = \lceil \log Cov^1(f) \rceil + 1$, and quantumly we have $NQcc(f) = \lceil \log nrank(f) \rceil + 1$. We now give a total function f with an exponential gap between $Ncc(f)$ and $NQcc(f)$. For $n > 1$, define f by

$$f(x,y) = 1 \text{ iff } |x \wedge y| \neq 1.$$

We first show that the quantum complexity $NQcc(f)$ is low.

THEOREM 3.5. *For the above f, we have $NQcc(f) \leq \lceil \log(n+1) \rceil + 1$.*

Proof. By Theorem 3.3, it suffices to prove $nrank(f) \leq n+1$. We will derive a low-rank nondeterministic matrix from the polynomial p of (2.1), using a technique from [43]. Let M_i be the matrix defined by $M_i(x,y) = 1$ if $x_i = y_i = 1$ and by $M_i(x,y) = 0$ otherwise. Notice that M_i has rank 1. Define a $2^n \times 2^n$ matrix M by

$$M(x,y) = \left(\sum_{i=1}^{n} M_i(x,y) \right) - 1.$$

Note that $M(x,y) = p(x \wedge y)$. Since p is a nondeterministic polynomial for the function which is 1 iff its input does not have weight 1, it can be seen that M is a nondeterministic matrix for f. Because M is the sum of $n+1$ rank-1 matrices, M itself has rank at most $n+1$. □

Now we show that the classical $Ncc(f)$ is high (both for f and its complement).

THEOREM 3.6. *For the above f, we have $Ncc(f) \in \Omega(n)$ and $Ncc(\overline{f}) \geq n-1$.*

Proof. Let $R_1, \ldots, R_k$ be a minimal 1-cover for f. We use the following result from [36, Example 3.22 and section 4.6], which is essentially due to Razborov [45].

> There exist sets $A, B \subseteq \{0,1\}^n \times \{0,1\}^n$ and a probability distribution $\mu : \{0,1\}^n \times \{0,1\}^n \to [0,1]$ such that all $(x,y) \in A$ have $|x \wedge y| = 0$, all $(x,y) \in B$ have $|x \wedge y| = 1$, $\mu(A) = 3/4$, and there are $\alpha, \delta > 0$ (independent of n) such that for all rectangles R, $\mu(R \cap B) \geq \alpha \cdot \mu(R \cap A) - 2^{-\delta n}$.

Since the R_i are 1-rectangles, they cannot contain elements from B. Hence $\mu(R_i \cap B) = 0$ and $\mu(R_i \cap A) \leq 2^{-\delta n}/\alpha$. However, since all elements of A are covered by the R_i, we have

$$\frac{3}{4} = \mu(A) = \mu\left(\bigcup_{i=1}^{k} (R_i \cap A) \right) \leq \sum_{i=1}^{k} \mu(R_i \cap A) \leq k \cdot \frac{2^{-\delta n}}{\alpha}.$$

Therefore, $Ncc(f) = \lceil \log k \rceil + 1 \geq \delta n + \log(3\alpha/4)$.

For the lower bound on $Ncc(\overline{f})$, consider the set $S = \{(x,y) \mid x_1 = y_1 = 1, x_i = \overline{y_i} \text{ for } i > 1\}$. This S contains 2^{n-1} elements, all of which are 1-inputs for $\overline{f}$. Note that if (x,y) and (x',y') are two elements from S, then $|x \wedge y'| > 1$ or $|x' \wedge y| > 1$, so a 1-rectangle for $\overline{f}$ can contain at most one element of S. This shows that a minimal 1-cover for $\overline{f}$ requires at least 2^{n-1} rectangles and $Ncc(\overline{f}) \geq n-1$. □

Another quantum-classical separation was obtained earlier by Massar et al. [37]. We include it for the sake of completeness. It shows that the nondeterministic complexity of the *non*equality problem is extremely low, in sharp contrast to the equality problem itself.

THEOREM 3.7 (see [37]). *For the nonequality problem on n bits, $NQcc(\mathrm{NE}) = 2$ versus $Ncc(\mathrm{NE}) = \log n + 1$.*

Proof. $Ncc(\mathrm{NE}) = \log n + 1$ is well known (see [36, Example 2.5]). Below we give the protocol for NE from [37].

Viewing her input x as a number $\in [0, 2^n - 1]$, Alice rotates a $|0\rangle$-qubit over an angle $x\pi/2^n$, obtaining a qubit $\cos(x\pi/2^n)|0\rangle + \sin(x\pi/2^n)|1\rangle$ which she sends to Bob. Bob rotates the qubit back over an angle $y\pi/2^n$, obtaining $\cos((x-y)\pi/2^n)|0\rangle + \sin((x-y)\pi/2^n)|1\rangle$. Bob now measures the qubit and sends back the observed bit. If $x = y$, then $\sin((x-y)\pi/2^n) = 0$, so Bob will always send 0. If $x \neq y$, then $\sin((x-y)\pi/2^n) \neq 0$, so Bob will send 1 with positive probability. □

In another direction, Klauck [34] showed that $NQcc(f)$ is in general incomparable to bounded-error quantum communication complexity: the latter may be exponentially larger or smaller, depending on f.

4. Future work. One of the main reasons for the usefulness of nondeterministic query and communication complexities in the classical case is the tight relation of these complexities with deterministic complexity.

In the query complexity (decision tree) setting, we have the well-known bound

$$\max\{N(f), N(\overline{f})\} \leq D(f) \leq N(f)N(\overline{f}).$$

We conjecture that something similar holds in the quantum case:

$$\max\left\{NQ(f), NQ(\overline{f})\right\} \leq Q_E(f) \leq D(f) \stackrel{?}{\leq} O(NQ(f)NQ(\overline{f})).$$

The ?-part is open and ties in with tightly embedding $NQ(f)$ and $ndeg(f)$ into the web of known relations between various complexity measures (section 2.6). This conjecture implies, for instance, $D(f) \in O(deg(f)^2)$, which would be close to optimal [42]. Similarly, it would imply $D(f) \in O(Q_0(f)^2)$, which would be close to optimal as well [18]. In both cases, the currently best relation has a fourth power instead of a square.

Similarly, for communication complexity, the following is known [36, section 2.11]:

$$\max\{Ncc(f), Ncc(\overline{f})\} \leq Dcc(f) \leq O(Ncc(f)Ncc(\overline{f})).$$

An analogous result might be true in the quantum setting, but we have been unable to prove it. So far, the best result in this direction is Klauck's observation that $Dcc(f) = O(Ncc(f)NQcc(\overline{f}))$ [33, Theorem 1].

Appendix. Comparison with alternative definitions. As mentioned in the introduction, three different definitions of nondeterministic quantum complexity are possible. We may consider the complexity of quantum algorithms that

1. output 1 iff given an appropriate *classical* certificate (and such certificates must exist iff $f(x) = 1$),
2. output 1 iff given an appropriate *quantum* certificate (and such certificates must exist iff $f(x) = 1$), or
3. output 1 with positive probability iff $f(x) = 1$.

The third definition is the one we adopted for this paper. Clearly definition 2 is at least as strong as definition 1 in the sense that the complexity of a function according to definition 2 will be less than or equal to the complexity according to definition 1. In fact, in the setting of query complexity, these two definitions are equivalent, because without loss of generality the certificate can be taken to be the purported input. See Aaronson [1] for some recent results about "quantum certificate (query) complexity."

Here we show that definition 3 is at least as strong as definition 2. We give the proof for the query complexity setting, but the same proof can be modified to work for communication complexity and other nonuniform settings as well. We then give an example in which the query complexity according to definition 3 is much less than according to definition 2. This shows that our $NQ(f)$ is in fact the most powerful definition of nondeterministic quantum query complexity.

We formalize definition 2 as follows. A *T-query quantum verifier* for f is a T-query quantum algorithm V together with a set $\mathcal{C}$ of m-qubit states, such that for all $x \in \{0,1\}^n$ we have (1) if $f(x) = 1$, then there is a $|\phi_x\rangle \in \mathcal{C}$ such that $V_x|\phi_x\rangle$ has

acceptance probability 1; and (2) if $f(x) = 0$, then $V_x|\phi\rangle$ has acceptance probability 0 for every $|\phi\rangle \in \mathcal{C}$. Informally, the set $\mathcal{C}$ contains all possible certificates: (1) for every 1-input, there is a verifiable 1-certificate in $\mathcal{C}$; and (2) for 0-inputs, there are not any. We do not put any constraints on $\mathcal{C}$. However, note that the definition implies that if $f(x) = 0$ for some x, then $\mathcal{C}$ cannot contain *all* m-qubit states; otherwise, $|\phi_x\rangle = V_x^{-1}|1\vec{0}\rangle$ would be a 1-certificate in $\mathcal{C}$ even for x with $f(x) = 0$.

We now prove that a T-query quantum verifier can be turned into a T-query nondeterministic quantum algorithm according to our third definition. This shows that the third definition is at least as powerful as the second. In fact, this even holds if we replace the acceptance probability 1 in clause (1) of the definition of a quantum verifier by just positive acceptance probability—in this case, both definitions are equivalent.

THEOREM A.1. *If there is a T-query quantum verifier V for f, then $NQ(f) \leq T$.*

Proof. The verifier V and the associated set $\mathcal{C}$ satisfy the following:

1. If $f(x) = 1$, then there is a $|\phi_x\rangle \in \mathcal{C}$ such that $V_x|\phi_x\rangle$ has acceptance probability 1.
2. If $f(x) = 0$, then $V_x|\phi\rangle$ has acceptance probability 0 for all $|\phi\rangle \in \mathcal{C}$.

Let $X_1 = \{z \mid f(z) = 1\}$. For each $z \in X_1$, choose one specific 1-certificate $|\phi_z\rangle \in \mathcal{C}$. Now let us consider some input x and see what happens if we run $V_x \otimes I$ (where I is the $2^n \times 2^n$ identity operation) on the $m + n$-qubit state

$$|\phi\rangle = \frac{1}{\sqrt{|X_1|}} \sum_{z \in X_1} |\phi_z\rangle|z\rangle.$$

V_x acts on only the first m qubits of $|\phi\rangle$; the $|z\rangle$-part remains unaffected. Therefore, running $V_x \otimes I$ on $|\phi\rangle$ gives the same acceptance probabilities as when we first randomly choose some $z \in X_1$ and then apply V_x to $|\phi_z\rangle$. In the case when $f(x) = 0$, this $V_x|\phi_z\rangle$ will have acceptance probability 0, so $(V_x \otimes I)|\phi\rangle$ will have acceptance probability 0 as well. In the case when the input x is such that $f(x) = 1$, the specific certificate $|\phi_z\rangle$ that we chose for this x will satisfy that $V_x|\phi_x\rangle$ has acceptance probability 1. However, then $(V_x \otimes I)|\phi\rangle$ has acceptance probability at least $1/|X_1| > 0$. Accordingly, $(V_x \otimes I)|\phi\rangle$ has positive acceptance probability iff $f(x) = 1$. By prefixing $V_x \otimes I$ with a unitary transformation that maps $|\vec{0}\rangle$ (of $m + n$ qubits) to $|\phi\rangle$, we have constructed a nondeterministic quantum algorithm for f with T queries. □

The above proof shows that our definition of $NQ(f)$ is at least as strong as the certificate-verifier definition. Could it be that both definitions are in fact equivalent (i.e., yield the same complexity)? The function we used in section 2.5 shows that this is not the case. Consider again

$$f(x) = 1 \text{ iff } |x| \neq 1.$$

It satisfies $NQ(f) = 1$. On the other hand, if we take a T-query verifier for f and fix the certificate for the all-0 input, we obtain a T-query algorithm that always outputs 1 on the all-0 input and that outputs 0 on all inputs of Hamming weight 1. The quantum search lower bounds [9, 6] immediately imply $T = \Omega(\sqrt{n})$. This shows that our definition of $NQ(f)$ is strictly more powerful than the certificate-verifying one.

Acknowledgments. Many thanks to Peter Høyer for stimulating discussions and for his permission to include the nondeterministic parts of our joint paper [29] in this paper. I also thank Scott Aaronson, Harry Buhrman, Richard Cleve, Wim

van Dam, Lance Fortnow, Hartmut Klauck, Peter Shor, and John Watrous for various helpful discussions, comments, and pointers to the literature. Thanks to the anonymous referees for suggesting some improvements.

REFERENCES

[1] S. Aaronson, *Quantum Certificate Complexity*, http://arxiv.org/abs/quant-ph/0210020 (2 Oct 2002); to appear in Proceedings of the 18th IEEE Conference on Computational Complexity, IEEE Computer Society, Los Alamitos, CA, 2003.

[2] L. M. Adleman, J. Demarrais, and M. A. Huang, *Quantum computability*, SIAM J. Comput., 26 (1997), pp. 1524–1540.

[3] N. Alon and J. H. Spencer, *The Probabilistic Method*, Wiley-Interscience, New York, 1992.

[4] A. Ambainis, *Quantum lower bounds by quantum arguments*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 636–643.

[5] A. Ambainis, L. Schulman, A. Ta-Shma, U. Vazirani, and A. Wigderson, *The quantum communication complexity of sampling*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 342–351.

[6] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf, *Quantum lower bounds by polynomials*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 352–361.

[7] N. de Beaudrap, R. Cleve, and J. Watrous, *Sharp quantum vs. classical query complexity separations*, Algorithmica, 34 (2002), pp. 449–461.

[8] R. Beigel, *The polynomial method in circuit complexity*, in Proceedings of the 8th IEEE Structure in Complexity Theory Conference, IEEE, New York, 1993, pp. 82–95.

[9] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, *Strengths and weaknesses of quantum computing*, SIAM J. Comput., 26 (1997), pp. 1510–1523.

[10] G. Brassard, *Personal communication via email*, Université de Montréal, Canada, 2000.

[11] G. Brassard, *Quantum Communication Complexity (A Survey)*, http://arxiv.org/abs/quant-ph/0101005 (1 Jan 2001).

[12] G. Brassard and P. Høyer, *An exact quantum polynomial-time algorithm for Simon's problem*, in Proceedings of the 5th Israeli Symposium on Theory of Computing and Systems, Ramat-Gan, Israel, 1997, pp. 12–23.

[13] G. Brassard, P. Høyer, and A. Tapp, *Quantum algorithm for the collision problem*, ACM SIGACT News (Cryptology Column), 28 (1997), pp. 14–19.

[14] G. Brassard, P. Høyer, and A. Tapp, *Quantum counting*, in Proceedings of the 25th ICALP, Lecture Notes in Comput. Sci. 1443, Springer-Verlag, New York, 1998, pp. 820–831.

[15] H. Buhrman, *Quantum computing and communication complexity*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 70 (2000), pp. 131–141.

[16] H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf, *Quantum fingerprinting*, Phys. Rev. Lett., 87 (2001), article 167902.

[17] H. Buhrman, R. Cleve, and A. Wigderson, *Quantum vs. classical communication and computation*, in Proceedings of the 30th ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 63–68.

[18] H. Buhrman, R. Cleve, R. de Wolf, and Ch. Zalka, *Bounds for small-error and zero-error quantum algorithms*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 358–368.

[19] H. Buhrman and R. de Wolf, *Communication complexity lower bounds by polynomials*, in Proceedings of the 16th IEEE Conference on Computational Complexity, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 120–130.

[20] H. Buhrman and R. de Wolf, *Complexity measures and decision tree complexity: A survey*, Theoret. Comput. Sci., 288 (2002), pp. 21–43.

[21] R. Cleve and H. Buhrman, *Substituting quantum entanglement for communication*, Phys. Rev. A (3), 56 (1997), pp. 1201–1204.

[22] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, Proc. Roy. Soc. London Ser. A, 439 (1992), pp. 553–558.

[23] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, *A limit on the speed of quantum computation in determining parity*, Phys. Rev. Lett., 81 (1998), pp. 5442–5444.

[24] S. Fenner, F. Green, S. Homer, and R. Pruim, *Determining acceptance possibility for a quantum computation is hard for the polynomial hierarchy*, in Proceedings of the 6th Italian Conference on Theoretical Computer Science, Prato, Italy, 1998, pp. 241–252.

[25] L. Fortnow and J. Rogers, *Complexity limitations on quantum computation*, J. Comput. System Sci., 59 (1999), pp. 240–252.
[26] L. K. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the 28th ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 212–219.
[27] H. van der Holst, L. Lovász, and A. Schrijver, *The Colin de Verdière graph parameter*, in Graph Theory and Combinatorial Biology, János Bolyai Mathematical Society, Budapest, 1999, pp. 29–85.
[28] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1985.
[29] P. Høyer and R. de Wolf, *Improved quantum communication complexity bounds for disjointness and equality*, in Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 2285, Springer-Verlag, New York, 2002, pp. 299–310.
[30] A. Kitaev, A. Shen, and M. Vyalyi, *Classical and Quantum Computation*, AMS, Providence, RI, 2002.
[31] A. Kitaev and J. Watrous, *Parallelization, amplification, and exponential time simulation of quantum interactive proof systems*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, ACM, New York, 2000, pp. 608–617.
[32] A. Y. Kitaev, *Quantum NP*, talk given at the 2nd Workshop on Algorithms in Quantum Information Processing, DePaul University, Chicago, 1999.
[33] H. Klauck, *Quantum communication complexity*, in Proceedings of the Workshop on Boolean Functions and Applications at the 27th International Colloquium on Automata, Languages and Programming, Geneva, Switzerland, 2000, pp. 241–252.
[34] H. Klauck, *Lower bounds for quantum communication complexity*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 288–297.
[35] I. Kremer, *Quantum Communication*, Master's thesis, Computer Science Department, Hebrew University, Jerusalem, Israel, 1995.
[36] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.
[37] S. Massar, D. Bacon, N. Cerf, and R. Cleve, *Classical simulation of quantum entanglement without local hidden variables*, Phys. Rev. A (3), 63 (2001), article 052305.
[38] C. Meinel and S. Waack, *The "log rank" conjecture for modular communication complexity*, in Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1046, Springer-Verlag, New York, 1996, pp. 619–630.
[39] M. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge, MA, 1968, 1988.
[40] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.
[41] N. Nisan, *CREW PRAMs and decision trees*, SIAM J. Comput., 20 (1991), pp. 999–1007.
[42] N. Nisan and M. Szegedy, *On the degree of Boolean functions as real polynomials*, Comput. Complexity, 4 (1994), pp. 301–313.
[43] N. Nisan and A. Wigderson, *On rank vs. communication complexity*, Combinatorica, 15 (1995), pp. 557–565.
[44] R. Raz, *Exponential separation of quantum and classical communication complexity*, in Proceedings of the 31st ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 358–367.
[45] A. Razborov, *On the distributional complexity of disjointness*, Theoret. Comput. Sci., 106 (1992), pp. 385–390.
[46] J. T. Schwartz, *Fast probabilistic algorithms for verification of polynomial identities*, J. ACM, 27 (1980), pp. 701–717.
[47] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.
[48] D. R. Simon, *On the power of quantum computation*, SIAM J. Comput., 26 (1997), pp. 1474–1483.
[49] A. Ta-Shma, *Classical versus quantum communication complexity*, ACM SIGACT News (Complexity Column 23), 30 (1999), pp. 25–34.
[50] R. de Wolf, *Characterization of non-deterministic quantum query and quantum communication complexity*, in Proceedings of the 15th IEEE Conference on Computational Complexity, IEEE Computer Society, Los Alamitos, CA, 2000, pp. 271–278.
[51] R. de Wolf, *Quantum communication and complexity*, Theoret. Comput. Sci., 287 (2002), pp. 337–353.

[52] T. Yamakami and A. C.-C. Yao, $NQP_C = co\text{-}C_{=}P$, Inform. Process. Lett., 71 (1999), pp. 63–69.

[53] A. C.-C. Yao, *Some complexity questions related to distributive computing*, in Proceedings of the 11th ACM Symposium on Theory of Computing, ACM, New York, 1979, pp. 209–213.

[54] A. C.-C. Yao, *Quantum circuit complexity*, in Proceedings of the 34th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1993, pp. 352–360.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 700–716

SEMI-ONLINE MAINTENANCE OF GEOMETRIC OPTIMA AND MEASURES*

TIMOTHY M. CHAN†

Abstract. We give the first nontrivial worst-case results for dynamic versions of various basic geometric optimization and measure problems under the semi-online model, where during the insertion of an object we are told when the object is to be deleted. Problems that we can solve with sublinear update time include the Hausdorff distance of two point sets, discrete 1-center, largest empty circle, convex hull volume in three dimensions, volume of the union of axis-parallel cubes, and minimum enclosing rectangle. The decision versions of the Hausdorff distance and discrete 1-center problems can be solved fully dynamically. Some applications are mentioned.

Key words. computational geometry, dynamic data structures

AMS subject classifications. 68U05, 68Q25, 68P05

PII. S0097539702404389

1. Introduction.

Problems in computational geometry that admit simple and efficient static solutions can often be significantly harder to solve in the *dynamic* setting, when data are inserted and deleted and answers have to be updated quickly. For example, the *width* of a planar n-point set is an easy-to-state quantity and can be computed by a "textbook" $O(n \log n)$ algorithm, but a data structure that can maintain the width under arbitrary point updates in a manner faster than recomputing from scratch had eluded researchers for years and was found only recently [8]. In this paper, we look at more standard geometric optimization and measure problems, study their worst-case complexities in the dynamic setting, and try to gain a better understanding into generally what types of problems admit nontrivial dynamization results.

The importance of dynamic computational geometry was realized long ago [10], and while there have been many fundamental results in the area, our current knowledge is still limited. Dynamic data structures for all kinds of problems reducible to range searching [1], including linear/convex programming, are known. A class of *decomposable* query problems [5] has been recognized as easy, for which simple general tricks are known. A useful technique has been devised to deal with problems of the form, "Which pair of objects minimizes a function?" [14]. Yet, simple nonconvex minimization problems, like the width or smallest enclosing rectangle, or max-min problems, like the Hausdorff distance (see below), defy solutions by all these methodologies.

Over a decade ago, Dobkin and Suri [11] introduced the *semi-online* model as a restricted form of dynamic computation. The model assumes that when an object is inserted, we are given the time at which the object is to be deleted. The model simultaneously generalizes the *incremental* case (when there are no deletions) and the *off-line* case (when the entire insertion/deletion sequence is known in advance).

*Received by the editors March 19, 2002; accepted for publication (in revised form) January 27, 2003; published electronically April 17, 2003. A preliminary version of this paper appeared in *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, 2002. This work was supported in part by an NSERC research grant.

http://www.siam.org/journals/sicomp/32-3/40438.html

†School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada (tmchan@uwaterloo.ca).

TABLE 1.1
Summary of results.

Semi-online update time		
Hausdorff distance (2-d)	$\widetilde{O}(n^{5/6})$	(Corollary 3.2)
Discrete 1-center (2-d)	$\widetilde{O}(n^{5/6})$	(Corollary 3.2)
Largest empty circle	$\widetilde{O}(n^{7/8})$	(Corollary 5.2)
Volume of 3-d convex hull	$\widetilde{O}(n^{7/8})$	(Corollary 5.3)
Volume of union of unit cubes	$\widetilde{O}(n^{1/2})$	(Theorem 6.1)
Minimum-perimeter/area rectangle	$\widetilde{O}(n^{1/2})$ / $\widetilde{O}(n^{5/6})$	(Corollary 7.2/7.3)
Fully dynamic update time (amortized)		
Hausdorff distance decision (2-d)	$\widetilde{O}(n^{1/2})$	(Corollary 4.2)
Discrete 1-center decision (2-d)	$\widetilde{O}(n^{1/2})$	(Corollary 4.2)
Largest discrete empty circle	$\widetilde{O}(1)$	(Corollary 4.4)

Considering the apparent difficulty in obtaining worst-case results under the original fully dynamic model, such a restriction is generally a reasonable starting point for research; it lets us obtain semi-dynamic algorithms that are provably efficient and still practically relevant. We adopt the semi-online model in this paper and initiate the study of problems of a broader scope and higher complexity than those originally considered by Dobkin and Suri themselves. (As we now know, most of the problems they considered can be solved fully dynamically by the techniques mentioned, notably Eppstein's work [14].)

Table 1.1 gives a list of problems familiar to computational geometers and the results obtained in this paper. Background to many of the problems can be found in texts such as [27]. Statically, each problem can be solved in $O(n \log n)$ time, by construction of a planar convex hull, three-dimensional convex hull (in particular, planar Voronoi diagram), or three-dimensional union of cubes. Dynamically, however, one can see plenty of challenges—we are unaware of nontrivial worst-case results for any of these problems, even in the incremental case!

Specifically, the first problem is to determine the Hausdorff distance $\max_{q \in Q} \min_{p \in P} d(p, q)$ for two given sets P and Q of n points in the plane, subject to semi-online updates to both P and Q, where $d(\cdot, \cdot)$ is the Euclidean metric; the Hausdorff distance is commonly used as a measure of resemblance between two images. Given planar point set P, the discrete 1-center problem asks for the smallest circle, centered at a point of P, that encloses P; this is a popular variant of the standard 1-center problem that cannot be solved by convex programming (in fact, Dobkin and Suri [11] raised this as an open question for semi-online algorithms). In contrast, the largest empty circle problem, another example in facility location, seeks the largest circle whose interior avoids P, with center inside a given region, say, a triangle Δ (note that there are different versions of the center constraint in the literature). The next two are measure problems in three dimensions and ask for the volume of the convex hull of n points and of the union of a set of n congruent axis-parallel cubes. The last (related to the width problem) is to find the smallest rectangle, in terms of perimeter or area, that encloses a planar point set; this "bounding box" is allowed to have arbitrary orientation. All these problems have practical applications. We obtain a sublinear semi-online algorithm for each, using $\widetilde{O}(n)$ space. (Throughout the paper, the $\widetilde{O}$ notation hides factors that are $o(n^{\varepsilon})$ for any fixed $\varepsilon > 0$.)

Along the way, we notice a few easier variants that admit sublinear fully dynamic

algorithms. These variants include the decision versions of the Hausdorff distance and discrete 1-center problems (decide whether the Hausdorff distance is less than a given value r, and decide whether there exists a circle of radius r, centered at a point of P, that encloses P), as well as the discrete version of the largest empty circle problem (find the largest circle, centered at a point of P, whose interior avoids P except the center).

Our semi-online algorithms all begin with a very simple strategy that was used recently by the author to tackle the width problem [8] and is outlined in the next section (Lemma 2.1). As one might guess, data structures for range searching [1] are exploited to obtain the sublinear time bounds, but in nontrivial (and at times, creative) ways. Most of our results are therefore theoretical. It is assumed that the input is in general position, without loss of generality, by known perturbation schemes.

We mention some "applications" in section 8, including improved time bounds for Klee's measure problem [26] in the case of four-dimensional unit hypercubes, and for the minimum-diameter spanning tree problem, which are of independent interest.

2. The strategy for semi-online dynamization. The most common dynamization strategy [5, 24, 25] is based on decomposing a set of objects into subsets, solving the problem on each subset, and combining the answers. An update affects only a small number of subsets and thus can be efficiently handled. Unfortunately, this simple approach is not viable for any of our problems, because they are not directly "decomposable"—there is no effective way to combine the answers when the set is arbitrarily decomposed into a large number of subsets.

Fortunately, another very simple (but lesser known) approach works for us, using a weaker form of decomposability based on dividing the given set into *two* subsets, one of which is kept small, as explained in the lemma below. The strategy was most recently used by the author [8] (building on a previous work by Eppstein [16]) to obtain a fully dynamic algorithm for planar width with $\widetilde{O}(n^{1/2})$ amortized time, by a variant of the lemma that handles arbitrary deletions (with $\alpha = \beta = 1$). A similar idea was also used in one example from Dobkin and Suri's paper [11].

LEMMA 2.1. *Consider a problem* Π *with the following property, where* $\alpha \geq 1$ *and* $0 < \beta \leq 1$ *are constants: we can preprocess a set* S *of* n *objects into a data structure in* $\widetilde{O}(n)$ *time and space such that, given any additional set* S' *of* b *objects, we can solve* Π *on the set* $S \cup S'$ (block query) *in* $\widetilde{O}(b^\alpha n^{1-\beta})$ *time. Then we can solve* Π *on a set of* n *objects under semi-online updates in* $\widetilde{O}(n^{1-\beta/(1+\alpha)})$ *time per update, using* $\widetilde{O}(n)$ *space.*

Proof. We store most of the objects in a static set S, preprocessed in the given data structure, and put the rest in an auxiliary list S'. Insertions and deletions are done directly to S', but after every b updates, we reset S' to hold the objects with the b smallest deletion times and S to hold all other objects (this ensures that any object to be deleted in the next round of b updates is indeed in S', not S). The data structure for S has to be rebuilt, in $\widetilde{O}(n)$ time, for every b updates. At any time, S' has $O(b)$ size, so the solution can be computed by a block query in $\widetilde{O}(b^\alpha n^{1-\beta})$ time. The total time for n updates is therefore

$$\widetilde{O}((n/b) \cdot n \,+\, n \cdot (b^\alpha n^{1-\beta})) \;=\; \widetilde{O}(n^{2-\beta/(1+\alpha)}) \tag{2.1}$$

if we set the parameter $b \approx n^{\beta/(1+\alpha)}$. This proves an amortized time bound of $\widetilde{O}(n^{1-\beta/(1+\alpha)})$.

If the application insists on a *worst-case* time bound, a well-known modification (e.g., see [24, 25]) is required: spread the work of rebuilding the data structure for S evenly over the next $b/2$ updates. The data structure for S is available for the jth update whenever $j \bmod b \geq b/2$. A similar "shifted" version of S and S' can deal with the other case $j \bmod b < b/2$. □

All our efforts are now directed to demonstrating that our problems obey the requirement of the lemma. This task isn't necessarily straightforward, as we will see when we examine each problem in detail.

On one occasion (in a higher-dimensional application), we find the following modification of the lemma useful.

LEMMA 2.2. *Lemma* 2.1 *still holds if we are unable to preprocess S from scratch in $\widetilde{O}(n)$ time but can create a copy of the data structure for $S \cup S'$* (block insertion)*, given the data structure for S, within $\widetilde{O}(n + b^{\alpha} n^{1-\beta})$ time.*

Proof. This is similar to the previous proof, except that we will use multiple levels of decomposition (increasing space by a logarithmic factor). Specifically, let $\ell = \lceil \log_2(n/b) \rceil$. For each $i = 1, \ldots, \ell$, we maintain a partition of the objects into a set S_i, stored in a data structure, and an auxiliary list S'_i: insertions and deletions are done directly to S'_i, but after every $n/2^{i+1}$ updates, we reset S'_i to hold all objects with the $n/2^i$ smallest deletion times and S_i to the complement of S'_i. The size of S'_i is always $O(n/2^i)$.

After every $n/2^{i+1}$ updates, we need to rebuild the data structure for S_i. We know that $S_{i-1} \subseteq S_i$ (because at any time, S'_{i-1} must contain all objects with the $n/2^i$ smallest deletion times). Thus, a data structure for S_i can be created by copying the structure for S_{i-1} and inserting $S'_{i-1} \setminus S'_i$ in $O(\frac{n}{2^i b})$ blocks. The total number of block insertions over n updates is therefore $O(\sum_{i=1}^{\ell} 2^{i+1} \cdot \frac{n}{2^i b}) = \widetilde{O}(n/b)$. At any time, the solution can be computed by a block query to the data structure for S_ℓ, so we again arrive at the same expression (2.1).

If the application insists on a worst-case time bound, we can again spread the construction of the data structures over time, and for each i, maintain two shifted versions of S_i and S'_i to ensure that one of them is available at any given time. We omit the standard details. □

3. Optimizing over discrete points on a polytope. We consider first the problem of maintaining the Hausdorff distance $\max_{q \in Q} \min_{p \in P} d(p, q)$ dynamically. The difficulty here is the apparent necessity to know the nearest neighbor of *each* point $q \in Q$ to the point set P (unlike in the bichromatic closest pair problem studied by Eppstein [14] of the min-min type). As points are inserted to P, a large number of these nearest neighbors could change. The idea is that with Lemma 2.1 (and Lemma 2.2), we do not need to maintain the newest version of these nearest neighbors after every single update but only after a block of updates.

The method involves multilevel range searching tools. Although the description assumes familiarity with these tools, it is conceptually not complicated.

The method is quite general: we can combine the nearest neighbor distances by operators other than the maximum, and we can solve the problem in any fixed dimension. For this reason, we state a more general problem. Let d be a constant. Given a set H of hyperplanes in $\mathbb{R}^d$, define a mapping $\lambda_H : \mathbb{R}^{d-1} \to \mathbb{R}^d$ as follows: $\lambda_H(q)$ is the point obtained by lifting q vertically onto the lower envelope of H (in other words, the lowest intersection of H with the vertical line at q). Given a set of points $Q \subset \mathbb{R}^{d-1}$, we wish to maintain implicitly the set of points $\lambda_H(Q) = \{\lambda_H(q) \mid q \in Q\}$ (all lying on a polytope's boundary, as the title of this section suggests); actually,

we want the output to be $\square\lambda_H(Q)$ for some *decomposable* operator $\square$, satisfying the requirement that for any disjoint pair of sets S_1 and S_2, $\square S_1$ and $\square S_2$ can be combined to form $\square(S_1 \cup S_2)$ in constant time.

THEOREM 3.1. *Suppose that we can preprocess an n-point set $Q \subset \mathbb{R}^{d-1}$ in $\widetilde{O}(n)$ time so that $\square\lambda_h(Q)$ for any hyperplane h in $\mathbb{R}^d$ can be computed in $\widetilde{O}(n^{1-\gamma})$ time, with $\gamma \geq 1/d$. Then we can maintain $\square\lambda_H(Q)$ for a set Q of at most n points in $\mathbb{R}^{d-1}$ and a set H of at most n hyperplanes in $\mathbb{R}^d$ in $\widetilde{O}(n^{1-\frac{1}{d(\lfloor d/2\rfloor+1)}})$ time per semi-online update to Q and H.*

Proof. We show how to store Q and H so that $\square\lambda_{H\cup H'}(Q \cup Q')$ can be computed efficiently given small additional blocks Q' and H'.

Preprocess the hyperplanes H in $\widetilde{O}(n)$ time to support vertical ray shooting queries in $\widetilde{O}(n^{1-1/\lfloor d/2\rfloor})$ time [1]. For each $q \in Q$, compute $\lambda_H(q)$; for $d \leq 3$, this step takes $O(n \log n)$ time. In $\widetilde{O}(n)$ time, preprocess $\lambda_H(Q)$ for simplex range searching [1, 23] to form canonical subsets $\{Q_i\}_i$ of total size $\widetilde{O}(n)$ such that, given any query simplex $\Delta \subset \mathbb{R}^d$, we can retrieve all points $q \in Q$ with $\lambda_H(q) \in \Delta$ as a union of disjoint canonical subsets $\{Q_i\}_{i\in I}$ in $\widetilde{O}(n^{1-1/d})$ time, with $\sum_{i\in I} |Q_i|^{1-1/d} = \widetilde{O}(n^{1-1/d})$. Precompute $\square\lambda_H(Q_i)$ for every canonical subset Q_i. In addition, preprocess each Q_i as specified so that given any hyperplane h, $\square\lambda_h(Q_i)$ can be found in $\widetilde{O}(n^{1-\gamma})$ time; the total preprocessing time is $\widetilde{O}(n)$.

Given query sets Q' and H' of size b, construct the lower envelope of H' in $\widetilde{O}(b^{\lfloor d/2\rfloor})$ time and triangulate it into $\widetilde{O}(b^{\lfloor d/2\rfloor})$ $(d-1)$-simplices. Take each such simplex Δ, defined by hyperplane $h' \in H'$, say.

- Consider the points $q \in Q$ such that $\lambda_H(q)$ lies directly below Δ. By simplex range searching, these points can be partitioned into canonical subsets $\{Q_i\}_{i\in I}$. Take each Q_i. All points $q \in Q_i$ have $\lambda_{H\cup H'}(q) = \lambda_H(q)$; so combine the current answer with $\square\lambda_H(Q_i)$. The time required is $\widetilde{O}(n^{1-1/d})$.
- Consider the points $q \in Q$ such that $\lambda_H(q)$ lies directly above Δ. Again these points can be partitioned into canonical subsets $\{Q_i\}_{i\in I}$. Take each such Q_i. All points $q \in Q_i$ have $\lambda_{H\cup H'}(q) = \lambda_{h'}(q)$; so combine the current answer with $\square\lambda_{h'}(Q_i)$. The time required is $\widetilde{O}(\sum_{i\in I} |Q_i|^{1-\gamma}) = \widetilde{O}(n^{1-1/d})$.

Applying this process to all simplices requires $\widetilde{O}(b^{\lfloor d/2\rfloor} n^{1-1/d})$ time overall. The current answer is $\square\lambda_{H\cup H'}(Q)$. To get $\square\lambda_{H\cup H'}(Q \cup Q')$, compute $\lambda_{H\cup H'}(q')$ for each $q' \in Q'$ by vertical ray shooting on H and on H', in $\widetilde{O}(bn^{1-1/\lfloor d/2\rfloor})$ total time, and combine the current answer with $\square\lambda_{H\cup H'}(Q')$.

For $d \leq 3$, the preprocessing time is $\widetilde{O}(n)$, so Lemma 2.1 is applicable, with $\alpha = \lfloor d/2\rfloor$ and $\beta = 1/d$. For $d \geq 4$, we cannot afford to recompute $\lambda_H(Q)$ from scratch, so Lemma 2.2 is required: to build a new data structure for $Q \cup Q'$ and $H \cup H'$, compute $\lambda_{H\cup H'}(q)$ from $\lambda_H(q)$ for every $q \in Q$, by vertical ray shooting on H', in total time $\widetilde{O}(b^{\lfloor d/2\rfloor} + n)$; in addition, compute $\lambda_{H\cup H'}(q')$ for every $q' \in Q'$ in total time $\widetilde{O}(bn^{1-1/\lfloor d/2\rfloor})$; now, the rest of the new data structure can be built from scratch in $\widetilde{O}(n)$ time. ∎

Note that we have used simplex range searching only for point sets *in convex position*. If it is possible to improve the range searching results under this special case (for example, it is not difficult in two dimensions), the bound in the theorem would be improved as well.

The Hausdorff distance is just a special case of the above problem in one dimension higher by a standard transformation. The discrete 1-center problem can similarly be

solved, as it asks for a similar quantity $\min_{q\in P}\max_{p\in P} d(p,q)$. In a forthcoming application, we encounter an additively weighted variant that seeks $\max_{q\in Q}\min_{p\in P}[d(p,q)+w_q]$ or $\min_{q\in P}\max_{p\in P}[d(p,q)+w_q]$, given weights w_q; this variant can be solved similarly as well.

COROLLARY 3.2. *In $\mathbb{R}^d$, we can maintain the Hausdorff distance of two sets of at most n points and the discrete 1-center of a set of n points (possibly with additive weights) in $\widetilde{O}(n^{1-\frac{1}{(d+1)(\lceil d/2\rceil+1)}})$ time per semi-online update.*

Proof. For the unweighted Hausdorff distance problem, transform each point $p=(a_1,\ldots,a_d)$ in P to a $(d+1)$-dimensional hyperplane h in H with equation $\{(x_1,\ldots,x_{d+1}) \mid x_{d+1}=a_1^2+\cdots+a_d^2-2a_1x_1-\cdots-2a_dx_d\}$. The points in $\lambda_H(Q)$ correspond to the nearest neighbors of the points in Q to set P. The operator $\square$ takes the maximum of $x_{d+1}+x_1^2+\cdots+x_d^2$ (the actual nearest neighbor distance squared) over the points. The requirement in the theorem (finding $\square\lambda_h(Q)$ for a given hyperplane h) translates to finding the largest distance of Q to a given point p, which can be done by farthest neighbor queries with $\gamma=1/\lceil d/2\rceil$ [1].

In the weighted case, the operator $\square$ should instead return the maximum of $\sqrt{x_{d+1}+x_1^2+\cdots+x_d^2}+w$, where w is the weight of the point $q=(x_1,\ldots,x_d)$. The requirement translates to finding weighted farthest neighbors: more precisely, given query point $(a_1,\ldots,a_d)$, find the maximum of

$$\sqrt{a_1^2+\cdots+a_d^2-2a_1x_1-\cdots-2a_dx_d+x_1^2+\cdots+x_d^2}+w$$

over a set of $O(n)$ tuples $(x_1,\ldots,x_d,w)$. We can compare this maximum with any value b by halfspace range search in $d+2$ dimensions: given $(a_1,\ldots,a_d,b)$, find a tuple that satisfies $w\ge b$ or $[a_1^2+\cdots+a_d^2-b^2]-2a_1x_1-\cdots-2a_dx_d+2bw+[x_1^2+\cdots+x_d^2-w^2]\ge 0$. This gives a data structure with $\gamma=1/(\lfloor d/2\rfloor+1)$ for the decision query problem and, by parametric or randomized search, for the maximization query problem as well [1]. ∎

By linearization or the use of lower envelopes of surfaces and semi-algebraic range searching [1], sublinear results can also be obtained for other metric $d(\cdot,\cdot)$ with constant description complexity. For Hausdorff distances under the L_∞ metric, much simplification is possible, since orthogonal range searching [1, 27] replaces simplex/halfspace range searching, and an L_∞-Voronoi diagram [6] replaces the lower envelope; the time bound reduces to $O(n^{1-\frac{1}{\lceil d/2\rceil+1}}\operatorname{polylog} n)$.

4. Some easier variants. Next, we give faster algorithms for the *decision* versions of the Hausdorff distance and discrete 1-center problem. The algorithms are in fact fully dynamic and are obtained by directly modifying known range searching structures (more specifically, by augmenting a standard partition tree to store two extra numbers at each node). The idea is actually to generalize the problem of *testing* whether each point is above the lower envelope, to *counting* the number of hyperplanes below each point.

To state the generalized problem, define the mapping $c_H:\mathbb{R}^d\to\mathbb{N}$, where $c_H(q)$ is the number of hyperplanes of H that lie below q. Implicitly, we wish to maintain the multiset of numbers $c_H(Q)=\{c_H(q) \mid q\in Q\}$ for a given set of points $Q\subset\mathbb{R}^d$; more precisely, we want to output $\square c_H(Q)$, where the operator $\square$ is assumed to be decomposable and furthermore satisfy the property that for any set S of numbers and a number j, $\square(S+j)$ can be computed from $\square S$ in constant time (where $S+j=\{i+j \mid i\in S\}$).

THEOREM 4.1. *We can maintain $\Box c_H(Q)$ for a set Q of at most n points and a set H of at most n hyperplanes in $\mathbb{R}^d$ in $\widetilde{O}(n^{1-1/d})$ amortized time per update to Q and H fully dynamically, using $O(n)$ space.*

Proof. Assume that the size of H is m instead. Store the dual points of H in a dynamic data structure to support simplex range counting queries in $\widetilde{O}(m^{1-1/d})$ time and updates in $\widetilde{O}(1)$ amortized time [1]. The notation H_Δ denotes the subset of all hyperplanes of H crossing a given simplex Δ.

Matoušek's partition theorem [1, 21] asserts that any set Q of n points can be partitioned into $O(r)$ subsets $\{Q_i\}_i$, each of size at most n/r and enclosed in a simplex Δ_i, with the property that every hyperplane crosses $O(r^{1-1/d})$ of these simplices. We choose r to be a constant here; the construction time is then linear. Assuming that Q itself is enclosed in a simplex Δ, we can ensure that the Δ_i's are all inside Δ (by intersecting with Δ and retriangulating).

Let c_{Δ_i} be the number of hyperplanes in H_Δ that lie completely below Δ_i. Note that the duals of all hyperplanes intersecting Δ and below Δ_i form cells in a hyperplane arrangement of constant size. Therefore, we can compute c_{Δ_i} by a constant number of simplex range counting queries in $\widetilde{O}(m^{1-1/d})$ time.

Our data structure for (Q, Δ) consists of recursively constructed structures for $\{(Q_i, \Delta_i)\}_i$, together with the numbers $\{c_{\Delta_i}\}_i$ and the answer $\Box c_{H_\Delta}(Q)$.

Knowing the subanswers $\Box c_{H_{\Delta_i}}(Q_i)$, we can compute the answer $\Box c_{H_\Delta}(Q)$ as follows. Take each Q_i. All points $q \in Q_i$ have $c_{H_\Delta}(q) = c_{H_{\Delta_i}}(q) + c_{\Delta_i}$; so combine the current answer with $\Box(c_{H_{\Delta_i}}(Q_i) + c_{\Delta_i})$. Repeating for all Q_i's yields the desired answer in constant ($O(r)$) time. By evaluating answers bottom-up, we can thus preprocess our data structure in time $\widetilde{O}(nm^{1-1/d})$.

To insert a hyperplane h to H_Δ, we can first increment the count c_{Δ_i} for each simplex Δ_i completely above h, then recursively insert h to H_{Δ_i} for each simplex Δ_i crossed by h, and finally recompute the answer $\Box c_{H_\Delta}(Q)$ from the subanswers $\Box c_{H_{\Delta_i}}(Q_i)$ in the manner described above. To delete h from H_Δ, we proceed similarly, decrementing the counts c_{Δ_i} instead. The recurrence for the insertion/deletion time is $t(n) = O(r^{1-1/d})t(n/r) + O(r)$, which solves to $t(n) = O(n^{1-1/d+\varepsilon})$ for an arbitrarily small $\varepsilon > 0$ if r is made arbitrarily large.

To delete a point q from Q, we recursively delete q from the subset Q_i that contains it, and then recompute the answer $\Box c_{H_\Delta}(Q)$ from the subanswers as above. The required time is only $O(\log n)$.

Initially Δ can be set to $\mathbb{R}^d$. We have thus obtained a data structure that maintains the value $\Box c_H(Q)$, supports updates to H in $\widetilde{O}(n^{1-1/d})$ time, can be preprocessed for any given Q in $T(n) = \widetilde{O}(nm^{1-1/d})$ time, and supports deletions from Q in $D(n) = O(\log n)$ time.

It remains to handle insertions to Q. For this, we apply a well-known general technique of Bentley and Saxe [5, 24, 25], which decomposes Q into logarithmically many deletion-only subsets (recall that the operator $\Box$ is decomposable) and transforms any data structure with preprocessing time $T(n)$ and deletion time $D(n)$ into a data structure that handles arbitrary updates to Q in amortized time $O((T(n)/n)\log n + D(n)) = \widetilde{O}(m^{1-1/d})$. As can be verified, this transformation preserves our ability to perform insertions and deletions to H, with the update time for H increased by a logarithmic factor, which is still $\widetilde{O}(n^{1-1/d})$. ∎

COROLLARY 4.2. *Given at most n points and at most n balls in $\mathbb{R}^d$, we can determine the ball that contains the least/most points in $\widetilde{O}(n^{1-\frac{1}{d+1}})$ amortized time fully dynamically. In $\mathbb{R}^d$, we can compare the Hausdorff distance of two sets of at*

most n points or the discrete 1-center radius of a set of n points with a fixed value r in $\widetilde{O}(n^{1-1/d})$ amortized time fully dynamically.

Proof. Transform each point to a $(d+1)$-dimensional hyperplane as in the proof of Corollary 4.2, transform each ball with center $(x_1, \ldots, x_d)$ and radius r to a $(d+1)$-dimensional point $(x_1, \ldots, x_d, r^2 - x_1^2 - \cdots - x_d^2)$, and take $\square$ to mean maximum/minimum. The second statement follows from the first: the slight improvement in the time bound is due to the fact that the balls have equal radius; here, the lifted points in Q lie on a common d-dimensional surface in $\mathbb{R}^{d+1}$, and the duals of the hyperplanes in H also lie on a common d-dimensional surface, so Agarwal and Matoušek's improved partition theorem [2] is applicable. □

Returning to the exact Hausdorff distance problem, we quickly mention a straightforward but fast algorithm for the special case when we expect only a small number of nearest neighbors to change in an update. This result is not surprising, but it makes us appreciate our earlier worst-case algorithms better. Still, the special-case algorithm would be interesting in applications where the update sequence is "random" (e.g., see [15]). The subsequent corollary mentions one particular consequence involving the *all-nearest-neighbors graph*, which connects each point $p \in P$ to its nearest neighbor in $P \setminus p$.

THEOREM 4.3. *We can maintain $\lambda_H(Q)$ for a set Q of at most n points in $\mathbb{R}^{d-1}$ and a set H of at most n hyperplanes in $\mathbb{R}^d$ in $\widetilde{O}(kn^{1-\frac{2}{\lfloor d/2 \rfloor + 1}})$ amortized time fully dynamically, with $\widetilde{O}(n^{2-\frac{2}{\lfloor d/2 \rfloor + 1}})$ space, where k is the number of changes to $\lambda_H(Q)$.*

Proof. Store H in a dynamic data structure for vertical ray shooting with $\widetilde{O}(n^{1-\frac{2}{\lfloor d/2 \rfloor + 1}})$ query and amortized update time [1]. Store $\lambda_H(Q)$ in a dynamic data structure for halfspace range reporting with $\widetilde{O}(n^{1-\frac{2}{\lfloor d/2 \rfloor + 1}} + A)$ query time (A is the answer size) and $\widetilde{O}(n^{1-\frac{2}{\lfloor d/2 \rfloor + 1}})$ amortized update time [1, 3]. The point set $\lambda_H(Q)$ itself can be maintained as follows: when a point q is inserted/deleted, simply insert/delete $\lambda_H(q)$, computable by vertical ray shooting; when a hyperplane h is inserted to H, find all k points q with $\lambda_H(q)$ above h by a halfspace range reporting query and reset each such $\lambda_H(q)$ to $\lambda_h(q)$; when h is deleted, retrieve all k points q with $\lambda_H(q)$ set to $\lambda_h(q)$ and recompute each such $\lambda_H(q)$ by vertical ray shooting. □

COROLLARY 4.4. *In $\mathbb{R}^d$, we can maintain the nearest neighbor of each point in one set of at most n points to another set of at most n points in $\widetilde{O}(kn^{1-\frac{2}{\lceil d/2 \rceil + 1}})$ amortized time fully dynamically, where k is the number of changes to the nearest neighbors. We can maintain the all-nearest-neighbors graph of an n-point set in $\widetilde{O}(n^{1-\frac{2}{\lceil d/2 \rceil + 1}})$ amortized time. We can maintain the largest discrete empty ball in the same time.*

Proof. The first statement is immediate by the standard transformation. The second statement can be obtained by a monochromatic variant of the algorithm, with the following well-known observation [28]: the degree of the all-nearest-neighbors graph is bounded by a constant, and thus the number of changes to the graph is $k = O(1)$ for every update. The largest distance in this graph is the radius of the largest discrete empty ball. □

The maintenance of the all-nearest-neighbors graph was raised by several researchers in connection with dynamic closest pairs [28], but considering that a dynamic all-nearest-neighbors algorithm can indirectly be used to answer nearest neighbor queries (by repeatedly inserting and then deleting the query point), the above bound is probably close to optimal.

The usual tricks could perhaps make the amortized bounds worst-case.

5. Optimizing over vertices of a 3-polytope. To solve problems like the largest empty circle (the original continuous version), we need to optimize a function over *all* points on a lower envelope rather than just a discrete set of points. Dynamization appears even harder. We may infer from the application that the optimum must be located at a vertex of the polytope (since we are actually maximizing a convex function), but the polytope, and thus its set of vertices, can change drastically in the worst case as hyperplanes are inserted and deleted. (Minimizing a convex function over a polytope is of course convex programming, but maximizing a convex function seems to require examining every vertex.)

We cannot afford to maintain the polytope explicitly after every update, so the idea is again to invoke Lemma 2.1 and use only a static structure, periodically rebuilt after a block of updates. The structure this time is more involved, as it needs to support queries on not just a set of points but a set of facial features (such as line segments) that come from the polytope.

This approach of implicitly maintaining a polytope again works in any fixed dimension in theory, but we will focus on problems involving three-dimensional polytopes that have efficient static solutions. (When the dimension exceeds 3, the number of vertices may be $\Omega(n^2)$ or bigger.)

The setup is as before and assumes a decomposable operator $\square$. Given a set H of planes in $\mathbb{R}^3$, let V_H denote the set of vertices of the lower envelope of H. The objective is to maintain $\square V_H$.

THEOREM 5.1. *Suppose that we can preprocess a set E of n pairs of planes in $\mathbb{R}^3$ in $\widetilde{O}(n)$ time so that $\square\{h_1 \cap h_2 \cap h \mid (h_1, h_2) \in E\}$ for any plane h can be computed in $\widetilde{O}(n^{1-\gamma})$ time, with $\gamma \geq 1/4$. Then we can maintain $\square V_H$ for a set H of n planes in $\mathbb{R}^3$ in $\widetilde{O}(n^{7/8})$ time per semi-online update.*

Proof. We show how to store a static set H so that $\square V_{H \cup H'}$ can be computed quickly given a block H' of b planes.

First construct the three-dimensional lower envelope of H in $O(n \log n)$ time [27] and preprocess it to support membership and ray shooting queries in $O(\log n)$ time [1]. In $\widetilde{O}(n)$ time, preprocess its $O(n)$ vertices V_H for three-dimensional simplex range searching [1] so that given any tetrahedron Δ, we can compute $\square(V_H \cap \Delta)$ in $\widetilde{O}(n^{2/3})$ time. Next, in $\widetilde{O}(n)$ time, preprocess the $O(n)$ edges E_H of the lower envelope for triangle-intersection queries so as to form canonical subsets $\{E_i\}_i$ of total size $\widetilde{O}(n)$ such that, given any triangle Δ, we can retrieve all edges (line segments) intersecting Δ as a union of disjoint canonical subsets $\{E_i\}_{i \in I}$ in $\widetilde{O}(n^{3/4})$ time, with $\sum_{i \in I} |E_i|^{3/4} = \widetilde{O}(n^{3/4})$—this involves multilevel range/intersection searching tools [1] (specifically, semi-algebraic range searching [2] in Plücker space; e.g., see [17, proof of Theorem 3.1], which examined a similar subproblem of quadrilaterial-intersection queries for rays). For yet another level, preprocess each canonical subset E_i as specified so that given any plane h intersecting every edge of E_i, $\square\{e \cap h \mid e \in E_i\}$ can be computed in $\widetilde{O}(n^{1-\gamma})$ time.

Given query set H', construct the lower envelope of H' (with vertex set $V_{H'}$ and edge set $E_{H'}$) and triangulate it into $O(b)$ triangles. Take each triangle Δ defined by plane $h' \in H'$, say.

- Vertices of V_H that lie directly below Δ are also vertices of $V_{H \cup H'}$, so combine the current answer with $\square\{v \in V_H \mid v \text{ directly below } \Delta\}$ by simplex range searching in $\widetilde{O}(n^{2/3})$ time.
- Consider the edges of E_H that intersect Δ. These edges can be partitioned

into canonical subsets $\{E_i\}_{i\in I}$. Take each such E_i. The intersection of the edges of E_i with h' are all vertices of $V_{H\cup H'}$, so combine the current answer with $\square\{e \cap h' \mid e \in E_i\}$. The time required is $\widetilde{O}(\sum_{i\in I} |E_i|^{1-\gamma}) = \widetilde{O}(n^{3/4})$.

Applying this process to all triangles requires $\widetilde{O}(bn^{3/4})$ time overall. So far, all vertices of $V_{H\cup H'}$ that are either vertices of V_H or intersections of edges of E_H with planes of H' have been accounted for. We can also take each edge $e' \in E_{H'}$, determine the at most two vertices of the intersection with the lower envelope of H, by ray shooting, and combine the answer with these vertices. We can further take each vertex of $V_{H'}$ that lies below the lower envelope of H and combine the answer with such vertices. The additional time is $O(b \log n)$, and the end result is $\square V_{H\cup H'}$.

The conclusion now follows from Lemma 2.1 with $\alpha = 1$ and $\beta = 1/4$. ∎

COROLLARY 5.2. *We can maintain the largest empty circle of an n-point set in $\mathbb{R}^2$, with center restricted inside any given triangle Δ, in $\widetilde{O}(n^{7/8})$ time per semi-online update.*

Proof. Apply the same transformation as in the proof of Corollary 4.2 to obtain n planes in $\mathbb{R}^3$. Add three nearly vertical planes along the edges of Δ (to ensure that the n planes cannot be seen from below when outside Δ). The largest empty circle must be centered at a vertex of the lower envelope of these $n + 3$ planes H (usually a Voronoi vertex, except in boundary cases). The optimal radius is then $\square V_H$, with the same operator $\square$ to maximize $x_3 + x_1^2 + x_2^2$. The requirement in the theorem (finding $\square\{h_1 \cap h_2 \cap h \mid (h_1, h_2) \in E\}$) seeks, for a given query plane h, the maximum of $O(n)$ functions in terms of h, parametrizable in 2 variables (since our planes are defined as liftings of points in $\mathbb{R}^2$). These nasty-looking bivariate functions nonetheless have constant description complexity, and by known semi-algebraic range searching and vertical ray shooting techniques in three dimensions [1, 2], we can achieve $\gamma = 1/3$. ∎

COROLLARY 5.3. *We can maintain the volume of the convex hull of an n-point set in $\mathbb{R}^3$ in $\widetilde{O}(n^{7/8})$ time per semi-online update.*

Proof. Apply duality to transform each given point p to a plane p^* so that facets of the upper hull of the points correspond to vertices of the lower envelope of the planes. For a vertex v defined by planes p_1^*, p_2^*, and p_3^*, we associate with it the volume of the tetrahedron $op_1p_2p_3$ for some fixed point o sufficiently far below the convex hull. The operator $\square$ just sums the volumes associated with the vertices of the given set. Applying a similar process to the lower hull and taking the difference yields the volume of the convex hull.

The theorem requires the sum of the volumes of op_1p_2p over n given pairs (p_1, p_2) and a query point p. By inspecting the proof of the theorem, we can ensure that the generated pairs (p_1, p_2) are consistently oriented with respect to the query point p. The volume of op_1p_2p is then a linear function in p (defined by a standard determinant), so the sum is also a linear function in p. By precomputing the coefficients in linear time, we can answer the volume-sum query in constant time for any given p, so $\gamma = 1$. ∎

One can imagine more applications of Theorem 5.1: for example, counting the number of vertices of a convex hull in $\mathbb{R}^3$, or determining the smallest/largest-area Delaunay triangle in $\mathbb{R}^2$. (The surface area of the convex hull, though, behaves differently, as we have to sum square roots.)

6. Measuring a union of unit cubes. We can apply the same approach to implicitly maintain structures other than a 3-polytope. To illustrate the idea on a simple example, we now explore the union of n unit axis-parallel cubes in three dimensions,

a structure also known to have linear complexity [6]. $O(\sqrt{n}\log n)$ dynamic algorithms for measuring the union of squares (in fact, arbitrary axis-parallel rectangles) were previously known by simple variants of the k-d tree (see [26]).

THEOREM 6.1. *We can maintain the volume of the union of a collection C of n unit axis-parallel cubes in $\mathbb{R}^3$ in $\widetilde{O}(\sqrt{n})$ time per semi-online update.*

Proof. We show how to store C so that the volume of $\bigcup(C \cup C')$ can be computed quickly given an additional set C' of b unit cubes.

First compute $\bigcup C$ and decompose this three-dimensional region into a collection S of $O(n)$ disjoint boxes (boxes here are axis-parallel); this computation can be done in $O(n \log n)$ time, as shown in [9]. Preprocess S in $\widetilde{O}(n)$ time so that given any query box $q \subset \mathbb{R}^3$, we can determine the sum of the volumes of $\sigma \cap q$ over all $\sigma \in S$ in $\widetilde{O}(1)$ time; in a moment, we will see exactly how this can be accomplished by orthogonal range searching.

Given query set C', compute $\bigcup C'$ and decompose the complement of the region into a collection S' of $O(b)$ disjoint boxes. Take each box $\sigma' \in S'$. Perform the above query to find the total volume of $\sigma \cap \sigma'$ over all $\sigma \in S$ in $\widetilde{O}(1)$ time. Summing over all $\sigma' \in S'$ yields the total volume of $\bigcup C$ outside $\bigcup C'$. Adding the volume of $\bigcup C'$ itself yields the final answer. The overall time is $\widetilde{O}(b)$, so we can apply Lemma 2.1 with $\alpha = \beta = 1$.

It remains to describe how to sum volumes of $\sigma \cap q$ over all $\sigma \in S$ for a given query box q. Lift each box $\sigma \in \mathbb{R}^3$ to a point $\sigma^* \in \mathbb{R}^6$ by taking the six coordinates of the box. By orthogonal range searching [1, 27], we can retrieve all boxes in S whose liftings lie in one of the 2^6 quadrants at q^* as a union of $\widetilde{O}(1)$ canonical subsets in $\widetilde{O}(1)$ time, after $\widetilde{O}(n)$-time preprocessing into canonical subsets of $\widetilde{O}(n)$ total size. Now, boxes σ inside each such quadrant intersect q (if at all) in a consistent "pattern" so that the volume of $\sigma \cap q$ can be characterized by a polynomial function (degree 3 at most) on the coordinates of q. By precomputing the (constant number of) coefficients of the sum of these polynomial functions in linear time for each canonical subset, we can therefore compute the sum of the volume of $\sigma \cap q$ over all σ^* inside a quadrant at q^* in $\widetilde{O}(1)$ time for any given q. Summing over all quadrants answers the query. ∎

The running time above is actually $O(\sqrt{n}\,\mathrm{polylog}\, n)$. With more effort, we should be able to maintain the surface area of the union of n unit axis-parallel cubes in $\mathbb{R}^3$ or the area/perimeter of a union of n homothets of a constant-size convex polygon in $\mathbb{R}^2$ (the latter union also has linear complexity).

7. Optimizing with multiple convex polygons. We started with optimization problems dealing with the interaction of points and a polytope. We will close with a more complicated form of optimization, but in the two-dimensional plane, dealing with interaction between two or more convex polygons. The planar width problem is perhaps the simplest in this category and was addressed in a previous paper [8]. Its relative, the minimum enclosing rectangle problem, is the main subject of this section; unlike in [8], we are not able to obtain a fully dynamic algorithm because of the added complications.

We first state the abstract problem. Given a set of lines H, let V_H be as before (the set of vertices of the lower envelope), but abusing notation slightly, let λ_H be instead a mapping from $\mathbb{R}^2$ to $\mathbb{R}^2$: $\lambda_H(q)$ is the point lying on the lower envelope of H and the vertical line at q. Let s be a constant. Below, $\langle \lambda_{H_1}, \ldots, \lambda_{H_s} \rangle(Q)$ denotes the set of tuples of points $\{\langle \lambda_{H_1}(q), \ldots, \lambda_{H_s}(q) \rangle \mid q \in Q\}$. The objective is to maintain $\square\langle \lambda_{H_1}, \ldots, \lambda_{H_s} \rangle(V_{H_1} \cup \cdots \cup V_{H_s})$ for s sets of lines $H_1, \ldots, H_s$ and some decomposable

operator $\square$.

Notation. Given a proper subset of indices $J \subset \{1, \ldots, s\}$, the *$J$-selector* refers to the following operator $\otimes$: $\langle p_1, \ldots, p_s\rangle \otimes \langle q_1, \ldots, q_s\rangle = \langle r_1, \ldots, r_s\rangle$, where $r_j = p_j$ if $j \in J$, and $r_j = q_j$ if $j \notin J$. There are a constant number $(2^s - 1)$ of such selectors.

THEOREM 7.1. *Suppose that we can preprocess a set $S \subset (\mathbb{R}^2)^s$ of n tuples (each consisting of s points on a common vertical line) in $\widetilde{O}(n)$ time so that given any s lines $h_1, \ldots, h_s$ and selector $\otimes$, $\square\{\langle \lambda_{h_1}(p_1), \ldots, \lambda_{h_s}(p_1)\rangle \otimes \langle p_1, \ldots, p_s\rangle \mid \langle p_1, \ldots, p_s\rangle \in P\}$ can be computed in $\widetilde{O}(n^{1-\gamma})$ time. Then we can maintain $\square\langle \lambda_{H_1}, \ldots, \lambda_{H_s}\rangle(V_{H_1} \cup \cdots \cup V_{H_s})$ for given sets $H_1, \ldots, H_s$ of at most n lines in $\mathbb{R}^2$ in $\widetilde{O}(n^{1-\gamma/2})$ time per semi-online update to $H_1, \ldots, H_s$.*

Proof. We show how to store $H_1, \ldots, H_s$ so that $\square\langle \lambda_{H_1 \cup H'_1}, \ldots, \lambda_{H_s \cup H'_s}\rangle(V_{H_1 \cup H'_1} \cup \cdots \cup V_{H_s \cup H'_s})$ can be computed efficiently given additional blocks $H'_1, \ldots, H'_s$ of size b.

Compute the lower envelope of each H_j (a convex polygon) in $O(n \log n)$ time. For each $v \in V_{H_1} \cup \cdots \cup V_{H_s}$, find $\lambda_{H_j}(v)$ by binary search. Using a binary tree construction, we can form canonical subsets $\{V_i\}_i$ of total size $O(n \log n)$ such that for any j, we can retrieve all points $v \in V_{H_j}$ inside a vertical slab as a union of $O(\log n)$ disjoint canonical subsets. For each canonical subset V_i, preprocess the tuples $\{\langle \lambda_{H_1}(v), \ldots, \lambda_{H_s}(v)\rangle \mid v \in V_i\}$ in the specified data structure.

Given query sets $H'_1, \ldots, H'_s$, construct the lower envelope of each H'_j in $O(b \log b)$ time. Draw vertical lines at the vertices of these s envelopes. In addition, intersect each edge of these envelopes with each of the lower envelopes of $H_1, \ldots, H_s$, by binary search [27] in total $O(b \log n)$ time. Draw vertical lines at these intersections. As a result, the plane is divided into $O(b)$ vertical slabs. Take each vertical open slab σ. Within σ, the lower envelope of $H_j \cup H'_j$ coincides with either the lower envelope of H_j or a single line $h'_j \in H'_j$. So, $V_{H_j \cup H'_j} \cap \sigma$ is either $V_{H_j} \cap \sigma$ or $\emptyset$. Assume the former.

Partition the point set $V_{H_j} \cap \sigma$ into $O(\log n)$ canonical subsets. Take each canonical subset V_i. Now, $\lambda_{H_k \cup H'_k}(V_i)$ is either $\lambda_{H_k}(V_i)$ or $\lambda_{h'_k}(V_i)$ for each k. Thus, $\square\langle \lambda_{H_1 \cup H'_1}, \ldots, \lambda_{H_s \cup H'_s}\rangle(V_i)$ is merely $\square\{\langle \lambda_{h'_1}(v), \ldots, \lambda_{h'_s}(v)\rangle \otimes \langle \lambda_{H_1}(v), \ldots, \lambda_{H_s}(v)\rangle \mid v \in V_i\}$ for some selector $\otimes$, and so can be computed in $\widetilde{O}(n^{1-\gamma})$ time per canonical subset V_i.

We conclude that $\square\langle \lambda_{H_1 \cup H'_1}, \ldots, \lambda_{H_s \cup H'_s}\rangle((V_{H_1 \cup H'_1} \cup \cdots \cup V_{H_s \cup H'_s}) \cap \sigma)$ can be computed in $\widetilde{O}(n^{1-\gamma})$ time. Repeating this process over all slabs requires $\widetilde{O}(bn^{1-\gamma})$ time overall. In addition, each of the $O(b)$ vertical lines drawn may pass through a vertex v_0 of $V_{H_1 \cup H'_1} \cup \cdots \cup V_{H_s \cup H'_s}$. If so, compute $\langle \lambda_{H_1 \cup H'_1}(v_0), \ldots, \lambda_{H_s \cup H'_s}(v_0)\rangle$ by binary searches on the lower envelopes of H_j and H'_j and combine with the answer. The additional time is $O(b \log n)$, and the end result is $\square\langle \lambda_{H_1 \cup H'_1}, \ldots, \lambda_{H_s \cup H'_s}\rangle(V_{H_1 \cup H'_1} \cup \cdots \cup V_{H_s \cup H'_s})$.

The conclusion now follows from Lemma 2.1 with $\alpha = 1$ and $\beta = \gamma$. ∎

COROLLARY 7.2. *We can maintain the minimum-perimeter rectangle enclosing an n-point set $P \subset \mathbb{R}^2$ in $\widetilde{O}(n^{1/2})$ time per semi-online update.*

Proof. We represent the rectangle using five parameters $\xi, \eta_1, \ldots, \eta_4$:

$$\{(x, y) \mid \ -\eta_1 \le \xi x + y \le \eta_2, \ \ -\eta_3 \le x - \xi y \le \eta_4\}.$$

The perimeter is $2(\eta_1 + \cdots + \eta_4)/\sqrt{1 + \xi^2}$.

So, transform each point $(a, b) \in P$ to the following four lines: $\{(\xi, \eta_1) \mid \eta_1 = -\xi a - b\}$ in H_1, $\{(\xi, \eta_2) \mid \eta_2 = \xi a + b\}$ in H_2, $\{(\xi, \eta_3) \mid \eta_3 = -a + \xi b\}$ in H_3, and $\{(\xi, \eta_4) \mid \eta_4 = a - \xi b\}$ in H_4. Define the operator $\square$ to minimize $(\eta_1 + \cdots + \eta_4)/\sqrt{1 + \xi^2}$ over all tuples $\langle(\xi, \eta_1), \ldots, (\xi, \eta_4)\rangle$ of the given set. Our problem is equivalent to

determining $\Box\langle\lambda_{H_1},\ldots,\lambda_{H_4}\rangle(\mathbb{R}^2)$. Since one of four sides of the optimal rectangle must be defined by a convex hull edge of the original points (see the appendix), our problem reduces to finding $\Box\langle\lambda_{H_1},\ldots,\lambda_{H_4}\rangle(V_{H_1}\cup\cdots\cup V_{H_4})$, so Theorem 7.1 is applicable with $s=4$.

Now, given the selector's index set $J\subset\{1,\ldots,4\}$, the theorem requires a data structure to store a set S of n tuples so that given query lines $\{(x,y)\mid y=s_jx+t_j\}$ $(j=1,\ldots,4)$, we can find the tuple $\langle(\xi,\eta_1),\ldots,(\xi,\eta_4)\rangle\in S$ that maximizes

$$\frac{\sum_{j\in J}(s_j\xi+t_j)+\sum_{j\notin J}\eta_j}{\sqrt{1+\xi^2}} = \frac{\xi}{\sqrt{1+\xi^2}}\sum_{j\in J}s_j+\frac{1}{\sqrt{1+\xi^2}}\sum_{j\in J}t_j+\frac{\sum_{j\notin J}\eta_j}{\sqrt{1+\xi^2}}.$$

By storing the plane

$$\left\{(X,Y,Z)\mid Z=\frac{\xi}{\sqrt{1+\xi^2}}X+\frac{1}{\sqrt{1+\xi^2}}Y+\frac{\sum_{j\notin J}\eta_j}{\sqrt{1+\xi^2}}\right\}$$

associated with each tuple for vertical ray shooting in three dimensions [1, 27] (by constructing a three-dimensional convex hull and performing two-dimensional point location), we can achieve $\gamma=1$. □

COROLLARY 7.3. *We can maintain the minimum-area rectangle enclosing an n-point set $P\subset\mathbb{R}^2$ in $\widetilde{O}(n^{5/6})$ time per semi-online update.*

Proof. Proceed as in the previous proof, but with a different objective to minimize $(\eta_1+\eta_2)(\eta_3+\eta_4)/(1+\xi^2)$.

The data structuring requirement is now more complex. Again we want to preprocess n tuples of the form $\langle(\xi,\eta_1),\ldots,(\xi,\eta_4)\rangle$. Let $\{(x,y)\mid y=s_jx+t_j\}$ $(j=1,\ldots,4)$ be the query lines. We consider the following cases of selectors only, as all other cases are symmetric or trivial.

- $J=\{1\}$. We want to minimize

$$\frac{(s_1\xi+t_1+\eta_2)(\eta_3+\eta_4)}{1+\xi^2} = \frac{\xi(\eta_3+\eta_4)}{1+\xi^2}s_1+\frac{\eta_3+\eta_4}{1+\xi^2}t_1+\frac{\eta_2(\eta_3+\eta_4)}{1+\xi^2}.$$

  By storing the plane $\{(X,Y,Z)\mid Z=\frac{\xi(\eta_3+\eta_4)}{1+\xi^2}X+\frac{\eta_3+\eta_4}{1+\xi^2}Y+\frac{\eta_2(\eta_3+\eta_4)}{1+\xi^2}\}$ associated with each tuple for three-dimensional vertical ray shooting as before, we can handle this type of query in $\widetilde{O}(1)$ time.
- $J=\{1,2\}$. We want to minimize

$$\frac{(s_1\xi+t_1+s_2\xi+t_2)(\eta_3+\eta_4)}{1+\xi^2} = \frac{\xi(\eta_3+\eta_4)}{1+\xi^2}[s_1+s_2]+\frac{\eta_3+\eta_4}{1+\xi^2}[t_1+t_2].$$

  This case reduces to two-dimensional vertical ray shooting.
- $J=\{1,3\}$. This is the most involved case and we will use linearization here [1]. We want to minimize

$$\begin{aligned}&\frac{(s_1\xi+t_1+\eta_2)(s_3\xi+t_3+\eta_4)}{1+\xi^2}\\ &\quad=s_1s_3+\frac{1}{1+\xi^2}[t_1t_3-s_1s_3]+\frac{\xi}{1+\xi^2}[s_1t_3+s_3t_1]\\ &\qquad+\frac{\xi\eta_4}{1+\xi^2}s_1+\frac{\eta_4}{1+\xi^2}t_1+\frac{\xi\eta_2}{1+\xi^2}s_3+\frac{\eta_2}{1+\xi^2}t_3+\frac{\eta_2\eta_4}{1+\xi^2}.\end{aligned}$$

By storing the hyperplane $\{(X_1, \ldots, X_7) \mid X_7 = \frac{1}{1+\xi^2}X_1 + \frac{\xi}{1+\xi^2}X_2 + \frac{\xi\eta_4}{1+\xi^2}X_3 + \frac{\eta_4}{1+\xi^2}X_4 + \frac{\xi\eta_2}{1+\xi^2}X_5 + \frac{\eta_2}{1+\xi^2}X_6 + \frac{\eta_2\eta_4}{1+\xi^2}\}$ associated with each tuple for seven-dimensional vertical ray shooting, we can handle this type of queries in $\widetilde{O}(n^{2/3})$ time with $\widetilde{O}(n)$ preprocessing [1, 22].

- $J = \{1, 2, 3\}$. This is similar to the previous case. We want to minimize

$$\begin{aligned}\frac{(s_1\xi + t_1 + s_2\xi + t_2)(s_3\xi + t_3 + \eta_4)}{1+\xi^2} &= (s_1 + s_2)s_3 + \frac{1}{1+\xi^2}[(t_1 + t_2)t_3 - (s_1 + s_2)s_3] \\ &\quad + \frac{\xi}{1+\xi^2}[(s_1 + s_2)t_3 + s_3(t_1 + t_2)] + \frac{\xi\eta_4}{1+\xi^2}[s_1 + s_2] + \frac{\eta_4}{1+\xi^2}[t_1 + t_2].\end{aligned}$$

Again we can use vertical ray shooting, this time in four dimensions. Thus, we can achieve $\gamma = 1/3$. □

The running time for Corollary 7.2 is actually $O(\sqrt{n}\,\mathrm{polylog}\, n)$ (only elementary tools are used). We will leave the reader with the question of whether the same approach works for similar problems like the minimum enclosing equilateral triangle (or convex polygon of a fixed angle sequence).

8. Some consequences. Faster dynamic data structures can generally lead to faster implementations of static algorithms. We briefly indicate a few sample applications of our results to illustrate their importance.

The generalized discrete 2*-center problem.* Given an n-point set $P \subset \mathbb{R}^2$, we want to find two points $p_1, p_2 \in P$ to minimize $f(r_1, r_2)$ such that every point $q \in P$ is within radius r_1 of p_1 *or* within radius r_2 of p_2. Here, $f(\cdot, \cdot)$ is some constant-time computable function that is monotone increasing in both arguments. Agarwal, Sharir, and Welzl [4] studied the most basic version with $f(r_1, r_2) = \max\{r_1, r_2\}$ and gave an $\widetilde{O}(n^{4/3})$ algorithm, but other functions, such as $f(r_1, r_2) = r_1 + r_2$, are reasonable in some situations. (See [13] for results on the generalized version of the original 2-center problem, where p_1 and p_2 can be arbitrary points in $\mathbb{R}^2$.)

An exhaustive algorithm may try each point $p_1 \in P$, shrink a ball B centered at p_1 with a decreasing radius r_1, and compute $r_2 = \min_{q\in P} \max_{p\in P\setminus B} d(p, q)$. We need to try n possible radii r_1 for each of the n choices for p_1, and if we compute r_2 from scratch in $O(n \log n)$ time via the farthest-point Voronoi diagram each time, the total running time would be $O(n^3 \log n)$. By applying the method of Corollary 3.2 and noting that $P \setminus B$ is subjected to insertions only as B shrinks (the insertion sequence is actually off-line), we immediately obtain an improved time bound of $\widetilde{O}(n^{3-1/6})$. In any constant dimension d, the bound is $\widetilde{O}(n^{3-\frac{1}{(d+1)(\lceil d/2\rceil+1)}})$.

The minimum-diameter spanning tree. An interesting application of the generalized discrete 2-center problem was considered by Ho et al. [20]: given an n-point set $P \subset \mathbb{R}^d$, find a spanning tree that minimizes its *diameter* (i.e., the maximum distance over all pairs of points, where the "distance" between p and q refers to the sum of the edge lengths, measured in the Euclidean metric, along the path connecting p and q in the tree). As Ho et al. showed, the resulting tree turns out to have very low *link diameter* (every pair of points is connected by a path with at most three edges), and consequently, the problem reduces to an additively weighted version of the discrete 2-center problem, where the objective is to minimize $r_1 + r_2 + d(p_1, p_2)$.

We can use the same exhaustive-search algorithm, except that while considering a center candidate p_1, we assign each point $q \in P \setminus B$ an additive weight of $w_q =$

$d(p_1, q)$ and maintain $\min_{q \in P} \max_{p \in P \setminus B}[d(p, q) + w_q]$ instead (using the method of Corollary 3.2). This results in the first subcubic time bound ($\widetilde{O}(n^{3-\frac{1}{(d+1)(\lceil d/2 \rceil + 1)}})$) for the problem.

Greedy disk cover. Consider the following (NP-hard) geometric version of the set cover problem: given n points and n balls in $\mathbb{R}^d$, find the smallest number of balls that together cover all the points. Although various approximation algorithms have been proposed (e.g., see [7]), the most well known is perhaps the greedy algorithm (which has a logarithmic approximation factor): choose the ball that covers the most points, remove the ball and all points inside it, and repeat. The naive implementation would require $O(n)$ time per iteration, for a total time of $O(n^2)$. By applying the fully dynamic Corollary 4.2, we can reduce the running time of the greedy algorithm to $O(n^{2-\frac{1}{d+1}})$. For unit balls, the time reduces to $O(n^{2-1/d})$.

Klee's measure problem for unit hypercubes. Klee's measure problem in $\mathbb{R}^d$ seeks the volume of n axis-parallel boxes. The fastest algorithm known is due to Overmars and Yap from 1991 [26] and runs in $O(n^{d/2} \log n)$ time. The algorithm basically exploits an orthogonal binary space partition of the $(d-2)$-faces of the boxes, and because such partitions have a worst-case lower bound of size $\Theta(n^{d/2})$ [12], improvement appears difficult in general.

Better bounds for the special case of unit hypercubes are possible, however (e.g., see [18]), because the union of unit hypercubes has size $O(n^{\lfloor d/2 \rfloor})$ only [6]. For example, for $d = 3$, we can afford to construct the union explicitly [9] and therefore find the volume in $O(n \log n)$ time. In higher dimensions, assuming that there is an algorithm $\mathcal{A}_d$ to construct the union and decompose the interior/exterior into disjoint boxes in $\widetilde{O}(n^{\lfloor d/2 \rfloor})$ time (for $d > 3$, we are unable to find an explicit reference to such an algorithm), we can solve Klee's problem for unit hypercubes in $\widetilde{O}(n^{\lfloor d/2 \rfloor})$ time—an improvement over Overmars and Yap's bound for odd dimensions d.

An interesting question involves the case of unit hypercubes in *even* dimensions $d > 2$. By a standard space sweep, the four-dimensional Klee's problem reduces to the off-line maintenance of the volume of a dynamic three-dimensional union and, by Theorem 6.1, can therefore be solved in $\widetilde{O}(n^{3/2})$ time. This is surprising considering that the union itself may have quadratic size in $\mathbb{R}^4$. A similar approach works for higher even dimensions and yields a time bound of $\widetilde{O}(n^{\lceil d/2 \rceil - 1 + \frac{1}{\lceil d/2 \rceil}})$, assuming the existence of algorithm $\mathcal{A}_{d-1}$ (the simple calculations are left for the interested readers to verify).

Hypercubes of possibly different sizes can have unions of complexity $\Theta(n^{\lceil d/2 \rceil})$ [6]. Assuming the existence of an algorithm to construct and decompose the union in near-optimal time, we can also obtain a similar algorithm, in this case with running time $\widetilde{O}(n^{\lfloor d/2 \rfloor + \frac{1}{\lfloor d/2 \rfloor + 1}})$, which is an improvement in odd dimensions $d \geq 5$.

Of course, a solution for hypercubes implies a solution for *fat* axis-parallel boxes, where the edge lengths of a box differ by at most a constant factor. (Unlike Overmars and Yap's method, though, our method does not solve the related problem of computing the *depth* in an arrangement of boxes [18].)

9. Conclusion. We have shown that a number of basic geometric problems have nontrivial (*sublinear*) dynamization results under the *semi-online* model. More important than the specific results themselves, however, are our general strategies for attacking different categories of problems; these strategies can serve as helpful design models to tackle further problems in dynamic computational geometry.

Of course, the ultimate wish is to have *fully* dynamic, *polylogarithmic* algorithms,

but at present this appears to be beyond our grasp for any of the problems discussed here. We hope that our results will inspire more work in this challenging area.

Appendix. It is a well-known fact [19] that the minimum-area rectangle enclosing a set of planar points has one side touching an edge of the convex hull, but since we are unable to find a reference stating the analogous fact for the minimum-perimeter rectangle, we include a quick proof:

Parametrize the rectangle differently in terms of variables $\xi, \eta, \omega_1, \omega_2, \zeta$:

$$\{(x, y) \mid \ \omega_1 \le \xi x + \eta y \le \omega_1 + \zeta, \quad \omega_2 \le \eta x - \xi y \le \omega_2 + (1 - \zeta)\}.$$

The perimeter is $2/\sqrt{\xi^2 + \eta^2}$. The problem is to maximize $\xi^2 + \eta^2$ subject to the constraints that the n given points lie in the rectangle—these constraints are linear in $\xi, \eta, \omega_1, \omega_2, \zeta$. To finish, observe that the maximum of a convex function over a polytope must be located at a vertex, here defined by five five-dimensional bounding hyperplanes, two of which are associated with a common side of the rectangle. □

Acknowledgment. I thank Der-Tsai Lee for (accidentally) mentioning the minimum-diameter spanning tree application to me.

REFERENCES

[1] P. K. Agarwal and J. Erickson, *Geometric range searching and its relatives*, in Advances in Discrete and Computational Geometry, B. Chazelle, J. E. Goodman, and R. Pollack, eds., AMS, Providence, RI, 1999, pp. 1–56.

[2] P. K. Agarwal and J. Matoušek, *On range searching with semi-algebraic sets*, Discrete Comput. Geom., 11 (1994), pp. 393–416.

[3] P. K. Agarwal and J. Matoušek, *Dynamic half-space range reporting and its applications*, Algorithmica, 13 (1995), pp. 325–345.

[4] P. K. Agarwal, M. Sharir, and E. Welzl, *The discrete* 2*-center problem*, Discrete Comput. Geom., 20 (1998), pp. 287–305.

[5] J. Bentley and J. Saxe, *Decomposable searching problems* I: *Static-to-dynamic transformation*, J. Algorithms, 1 (1980), pp. 301–358.

[6] J.-D. Boissonnat, M. Sharir, B. Tagansky, and M. Yvinec, *Voronoi diagrams in higher dimensions under certain polyhedral distance functions*, Discrete Comput. Geom., 19 (1998), pp. 473–484.

[7] H. Brönnimann and M. T. Goodrich, *Almost optimal set covers in finite VC-dimension*, Discrete Comput. Geom., 14 (1995), pp. 263–279.

[8] T. M. Chan, *A fully dynamic algorithm for planar width*, Discrete Comput. Geom., to appear.

[9] L. P. Chew, D. Dor, A. Efrat, and K. Kedem, *Geometric pattern matching in d-dimensional space*, Discrete Comput. Geom., 21 (1999), pp. 257–274.

[10] Y.-J. Chiang and R. Tamassia, *Dynamic algorithms in computational geometry*, Proc. IEEE, 80 (1992), pp. 1412–1434.

[11] D. Dobkin and S. Suri, *Maintenance of geometric extrema*, J. ACM, 38 (1991), pp. 275–298.

[12] A. Dumitrescu, J. S. B. Mitchell, and M. Sharir, *Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles*, in Proceedings of the 17th ACM Symposium on Comput. Geom., Association for Computing Machinery, New York, 2001, pp. 141–150.

[13] D. Eppstein, *Dynamic three-dimensional linear programming*, ORSA J. Comput., 4 (1992), pp. 360–368.

[14] D. Eppstein, *Dynamic Euclidean minimum spanning trees and extrema of binary functions*, Discrete Comput. Geom., 13 (1995), pp. 111–122.

[15] D. Eppstein, *Average case analysis of dynamic geometric optimization*, Comput. Geom. Theory Appl., 6 (1996), pp. 45–68.

[16] D. Eppstein, *Incremental and decremental maintenance of planar width*, J. Algorithms, 37 (2000), pp. 570–577.

[17] D. Eppstein and J. Erickson, *Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions*, Discrete Comput. Geom., 22 (1999), pp. 569–592.

[18] J. Erickson, *Klee's Measure Problem*, available at http://compgeom.cs.uiuc.edu/~jeffe/open/klee.html, 1998.
[19] H. Freeman and R. Shapira, *Determining the minimum-area encasing rectangle for an arbitrary closed curve*, Commun. ACM, 18 (1975), pp. 409–413.
[20] J.-M. Ho, D. T. Lee, C.-H. Chang, and C. K. Wong, *Minimum diameter spanning trees and related problems*, SIAM J. Comput., 20 (1991), pp. 987–997.
[21] J. Matoušek, *Efficient partition trees*, Discrete Comput. Geom., 8 (1992), pp. 315–334.
[22] J. Matoušek, *Reporting points in halfspaces*, Comput. Geom. Theory Appl., 2 (1992), pp. 169–186.
[23] J. Matoušek, *Range searching with efficient hierarchical cuttings*, Discrete Comput. Geom., 10 (1993), pp. 157–182.
[24] K. Mehlhorn, *Data Structures and Algorithms* 3: *Multi-Dimensional Searching and Computational Geometry*, Springer-Verlag, Heidelberg, 1984.
[25] M. H. Overmars, *The Design of Dynamic Data Structures*, Lecture Notes in Comput. Sci. 156, Springer-Verlag, Heidelberg, 1983.
[26] M. H. Overmars and C.-K. Yap, *New upper bounds in Klee's measure problem*, SIAM J. Comput., 20 (1991), pp. 1034–1045.
[27] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
[28] M. Smid, *Closest-point problems in computational geometry*, in Handbook of Computational Geometry, J. Urrutia and J. Sack, eds., North-Holland, Amsterdam, 2000, pp. 877–935.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 717–735

IMPROVED BOUNDS FOR THE ONLINE SCHEDULING PROBLEM*

JOHN F. RUDIN III† AND R. CHANDRASEKARAN†

Abstract. The problem considered here is the same as the one discussed in [G. Galambos and G. J. Woeginger, eds., *SIAM J. Comput.,* 22 (1993), pp. 349–355]. It is an m-machine online scheduling problem in which we wish to minimize the competitive ratio for the makespan objective. In this paper, we show that $\sqrt{3}$ is a lower bound on this competitive ratio for $m = 4$. In particular, we show how to force a lower bound of $\sqrt{3} - \epsilon$ for any positive ϵ. This reduces the gap between the performance of known algorithms [S. Albers, in *Proceedings of the* 29*th Annual ACM Symposium on Theory of Computing,* ACM, New York, 1997, pp. 130–139] and the lower bound. The method used introduces an approach to building the task master's strategy.

Key words. scheduling, online, competitive ratio, lower bound

AMS subject classifications. Primary, 90B35; Secondary, 68M20

PII. S0097539702403438

1. Introduction.

We have m identical machines and a sequence of jobs to be processed on any one of these machines. Preemption is not allowed. We wish to minimize makespan. This is an online scenario: there are two players—a scheduler and a task master. The task master releases information on jobs (such as processing time) sequentially; the scheduler must schedule each job on a machine without knowing the future job information. These decisions are irrevocable. However, he knows the processing time of the current job before it is scheduled. At any time, the task master may stop issuing any further job. At this time, we calculate the ratio of the length of the schedule to the optimal offline schedule (which knows all of the data up front). The scheduler wants to minimize this ratio (known as *the competitive ratio*), and the task master attempts to make the scheduler look bad and therefore maximizes this ratio.

History. The problem of finding the optimal solution has been solved for two and three machines. In [8], Graham demonstrated that the greedy algorithm, now called list scheduling, had a competitive ratio of $2 - \frac{1}{m}$. The list scheduling algorithm always puts a new job on the machine with the lowest current load. This is an optimal algorithm for two or three machines, as was shown in [5]. For four or more machines, better algorithms have been found.

In 1989, Faigle, Kern, and Turan [5] showed a lower bound of $\frac{1+\sqrt{2}}{2} (\approx 1.7071)$ for all $m \geq 4$. For more than three machines, neither the best scheduling algorithm nor the competitive ratio is known. An upper bound that improved on Graham's greedy algorithm for four or more machines was discovered by Galambos and Woeginger [7] in 1993. Their algorithm had a competitive ratio of $2 - \frac{1}{m} - \epsilon(m)$ for some positive $\epsilon(m)$ (which tends to zero as $m \to \infty$). Bartal, Fiat, Karloff, and Vohra [2] provided an algorithm that was 1.978 competitive for all values of m. That bound was improved upon by Karger, Phillips, and Torng [10] with an algorithm for $m \geq 8$ that guarantees a competitive ratio of at most 1.945. This is a generalization of the earlier algorithm. In 1994 , Bartal, Karloff, and Rabani [3] improved the lower bound to 1.837 for

*Received by the editors March 5, 2002; accepted for publication (in revised form) April 23, 2002; published electronically April 17, 2003.

http://www.siam.org/journals/sicomp/32-3/40343.html

†Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083 (chandra @utdallas.edu).

TABLE 1.1

m	LB	UB
2	$\frac{3}{2}$	$\frac{3}{2}$
3	$\frac{5}{3}$	$\frac{5}{3}$
4	1.7310	1.7333
5	1.7462	1.7708
6	1.7730	1.8
7	1.7910	1.8229
∞	1.8520	1.9230

$m \geq 3454$. Albers [1] produced an algorithm which is 1.923 competitive. She also improved the lower bound to 1.852 for m a multiple of 40.

Table 1.1 shows the best currently known bounds, as given in Fiat and Woeginger [6]. Many of these bounds were taken from Chen, van Vliet, and Woeginger [4].

Our primary result in this paper is to show a task master strategy that forces a minimum ratio of $\sqrt{3} - \epsilon$ for any positive ϵ for four machines. By using this method we can also improve the lower bounds in general for $m \geq 4$, and this is taken up in a forthcoming paper.

2. General method. The method presented in this paper will improve the lower bounds by presenting sequences of jobs for a given competitive ratio $(1 + V)$ and m, such that the scheduler is forced to schedule in a specific manner. Specifically, successive *layers* of jobs will be produced, each containing m jobs, as shown in Figure 2.1. The layers are constructed such that if the scheduler puts any two of the jobs in the same layer on the same machine, that machine will instantly have a makespan greater than or equal to the current divisor (a lower bound on the offline schedule length) times the competitive ratio. Therefore, regardless of the scheduling algorithm, at the end of a series of layers, we know that one of the following conditions holds:

- each machine has one job from each layer, or
- the competitive ratio of $1 + V$ or more has already been achieved.

Such a sequence guarantees that at the end of each layer, every machine has a minimum load equal to the sum of the smallest job in each layer, or the target competitive ratio has already been achieved.

The *current layer* of jobs will be designated the *A-layer* and will be $A_1, A_2, \ldots, A_m$. The layer before it will be $B_1, B_2, \ldots, B_m$, and the layer before that will be $C_1, C_2, \ldots, C_m$. After the first job in a layer is determined, no job in that layer will ever be any smaller; that is, $A_1 \leq A_i$ for all i.

We define S as the lowest possible machine load, assuming that each machine has one job from each layer. The total S depends on the number of layers that have been scheduled, so we define

$$S_B = B_1 + C_1 + \cdots,$$
$$S_A = A_1 + B_1 + C_1 + \cdots. = A_1 + S_B = A_1 + B_1 + S_C,$$

etc.

Therefore, when an entire layer has been scheduled by any scheduling algorithm, either a competitive ratio greater or equal to $(1 + V)$ will have already been achieved or every machine will have a load of at least S. For the remainder of this paper, it will

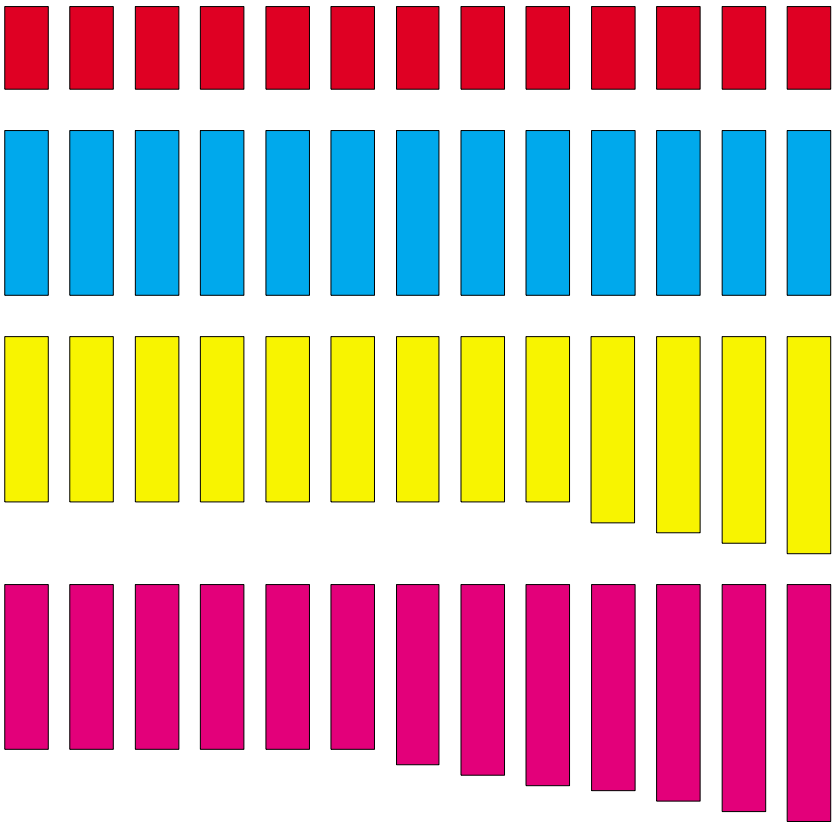


FIG. 2.1. *Typical sequence of jobs in the layering method.*

be assumed that the scheduler avoids the target competitive ratio as long as possible, and so it is assumed that at the end of each row, every machine has been given one job from each layer.

We also define the ratios

$$R_A = \frac{A_1}{S_A}, \quad R_B = \frac{B_1}{S_B},$$

etc.

The goal of the process is to produce a sequence of layers, each one reducing R, until $R = \frac{1}{2V}$. If $R = \frac{1}{2V}$, then, since $R_A = \frac{A_1}{S_A}$, we have

$$S_A = 2A_1V.$$

It is also necessary that all current jobs fit on an offline schedule on $m-1$ machines with loads no greater than $2A_1$. *This condition must be checked separately.* Then a final job of $2A_1$ is offered. No matter which machine it is scheduled on, that machine

will have a load of at least

$$2A_1 + S_A = 2A_1(1 + V).$$

Since the offline schedule for all jobs on m machines is $2A_1$, this will immediately force a competitive ratio of $1 + V$.

There will be two types of layers, each of m jobs, that we might use:

1. m jobs of equal size, i.e., $A_1 = A_2 = \cdots = A_i = \cdots = A_m$;
2. m jobs satisfying the relation $A_m \leq A_1 + 2C_1$; the offline makespan when A_m is being scheduled is exactly $A_1 + B_1 + 2C_1$.

In all cases, $A_1 \leq A_i$ for all i. They will be identically equal to A_1 as long as possible, becoming larger only when a larger value is needed to force the scheduler to schedule them on separate machines. The smallest load possible for a machine that already has a job from this layer is S_A. Therefore, A_i will be chosen so that a machine with load $S_A + A_i$ would force a competitive ratio of $1 + V$. For this reason, A_i will be $(1 + V)D_i - S_A$, where D_i is the divisor when A_i is to be scheduled. Since D_i is nondecreasing, it follows that $A_i \leq A_{i+1}$.

Consider the final situation, in which the next job of size J must force a competitive ratio of at least $1 + V$. We may assume, without loss of generality, that the divisor is 1 by scaling all jobs equally. Let S_A be the smallest total workload on any machine. Then the following statements are true:

1. $J \leq 1$ (no job can be greater than the divisor).
2. $S_A + J \geq 1 + V$.
3. Therefore, $S_A \geq V$.

The layering construction method presented assumes that a layer has just been completed as well as the fact that $J = 1$. Since the $m + 1$ jobs $A_1, \ldots, A_m$ plus J must all be scheduled on m machines with makespan less than or equal to 1, at least one of those jobs must be less than or equal to $\frac{1}{2}$. The smallest of these is A_1, so we assume that $A_1 = \frac{1}{2}$.

Therefore, for the final layer, the ratio $R_A = \frac{A_1}{S_A} \leq \frac{1}{2V}$. Achieving this ratio is a primary goal of the layering method.

2.1. Type-1 layers. Type-1 layers are layers of m identical jobs, that is, $A_1 = A_i$ for all i; and the scheduler is required to put each job of the layer on a separate machine, or he will immediately have caused the competitive ratio $1+V$ to have been reached. Let S_B be the smallest machine load prior to the placing of A_1. Then, by construction,

$$S_B = B_1 + C_1 + \cdots,$$
$$S_A = S_B + A_1.$$

Let $R = \frac{A}{S_A}$. Then, for a type-1 layer to force each A-job onto a distinct machine, we must have

$$A + S_A \geq (1 + V)D.$$

Since every machine has a load of at least S_A, it follows that $D \geq S_A$, with equality holding only if all previous layers are also type-1 layers. Therefore, A_1 must be chosen such that

$$A_1 + S_A \geq (1 + V)D \geq (1 + V)S_A.$$

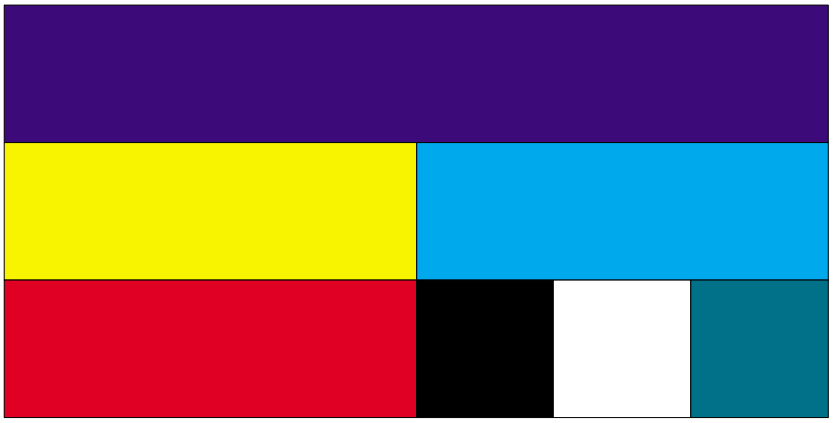


FIG. 2.2.

Hence

$$A_1 \geq VS_A,$$

and, therefore,

$$R = \frac{A_1}{S_A} \geq V.$$

By the previous result, R must be reduced to $\frac{1}{2V}$ by the end of the process. Therefore, type-1 layers can only achieve the desired competitive ratio if

$$V \leq R \leq \frac{1}{2V}.$$

This implies that $V^2 \leq \frac{1}{2}$, and hence $1 + V \leq 1 + \frac{1}{\sqrt{2}} \cong 1.7071$.

Using type-1 layers alone, Faigle, Kern, and Turan demonstrated that $\frac{3}{2}$ was the highest possible forced competitive ratio with two machines, and $\frac{5}{3}$ was the highest possible with three machines [5].

With two machines, a single layer of two jobs of size 1 is offered, followed by a job of size 2. The first two jobs are forced on separate machines to avoid the competitive ratio of 2, and then the final job forces a competitive ratio of $\frac{3}{2}$.

For three machines, two layers are needed. A layer of size $B = 1$ is followed by a layer of size $A = 3$, which must be scheduled on separate machines to avoid a competitive ratio of $\frac{7}{4}$. Since all jobs in type-1 layers are identical, no subscript is needed to distinguish them. At that point, every machine has a load of 4. When a final job of $2A$ is sent, then the machine it is scheduled on must have a makespan of 10. The offline schedule has makespan 6, as shown in Figure 2.2.

Therefore, a competitive ratio of $\frac{5}{3}$ is forced.

Since Graham's list scheduling algorithm can never be forced above $2 - \frac{1}{m}$, this is the highest that can be forced for $m = 2, 3$.

In 1989, Faigle, Kern, and Turan [5] showed a lower bound of $1 + \frac{\sqrt{2}}{2} (\cong 1.7071)$ for all $m \geq 4$. They used two type-1 layers. The first layer is $\frac{\sqrt{2}}{2} - \frac{1}{2} (\cong 0.2071)$. The

second layer is $\frac{1}{2}$; if two of these jobs are scheduled on the same machine, the ratio is

$$\frac{(\frac{\sqrt{2}}{2}+\frac{1}{2})}{\frac{\sqrt{2}}{2}} = 1+\frac{\sqrt{2}}{2}.$$

A final job of size 1 completes the sequence. As shown above, this is the highest that can be achieved with type-1 layers alone. To force higher ratios, we need type-2 layers.

2.2. Type-2 layers. A type-2 layer is a layer of m jobs, satisfying the relation

$$A_m \le A_1 + 2C_1,$$

and the divisor when A_m is being scheduled is exactly $A_1 + B_1 + 2C_1$. At the end of a layer, there are m A-layer jobs, each greater than or equal to A_1. It follows that if the ideal makespan is to be less than $2A_1$, then each of the m A-level jobs must be scheduled on different machines, so each machine contains at least A_1, from an A-level job, and unless it was a type-1 layer, one of the machines received $A_m > A_1$. Then, either one of the machines has two B-level jobs, or each machine has one of them. Therefore, the makespan is at least $\min[A_1 + 2B_1, A_m + B_1]$. Therefore, if

$$A_m \ge A_1 + B_1,$$

then the divisor is at least $A_1 + 2B_1$. For the divisor to go below this bound, each A-level job must be on a separate machine, and each B-level job must be as well. Therefore, to reduce the divisor further, we must enforce $A_m < A_1 + B_1$. By the same reasoning as before, if

$$A_m \ge A_1 + C_1,$$

then the competitive makespan is at least $A_1 + B_1 + 2C_1$. Type-2 layers are designed to force this competitive ratio. A type-2 layer is created by keeping the competitive divisor less than or equal to $A_1 + B_1 + 2C_1$ during the offering of the layer. This is always less than $A_1 + 2B_1$ or $2A_1$.

Since there are m separate jobs of size A_1 or greater, this bound requires m A-level jobs to be on the m machines for the competitive schedule. Since there are also m separate jobs of size B_1 or greater, these jobs must also each be on separate machines for the competitive schedule. Therefore, the machine with job $A_m (= A_1 + 2C_1)$ and some B-level job has a makespan of at least $A_1 + B_1 + 2C_1$. It is therefore sufficient to show a competitive schedule with makespan $A_1 + B_1 + 2C_1$ to prove that is the competitive divisor.

Example 1. As an example, consider the following series of jobs for $m = 4$ and $V = \frac{19}{11} = 1.727272$:

1	1	1	1
3	3	3	3
16	16	16	16
44	44	44	50
88			

The first three layers are type-1. They must be assigned across all four machines to avoid a competitive ratio of $\frac{7}{4}$. Therefore, at the end of the third row, each machine has a makespan of 20. As the first three numbers in the fourth layer are being assigned, the competitive divisor can be no more than 60, as shown by the following offline schedule: $[44, 16]; [44, 16]; [44, 16]; [16, 3, 3, 3, 3, 1, 1, 1, 1]$. It follows that the scheduler cannot allow any machine to exceed $60(\frac{19}{11})(= 103.636)$. *Note that the argument does not require proof that* 60 *is the length of the ideal schedule—merely that it is an upper bound on the length.*

Since each machine already has a load of 20, the scheduler cannot put any two of the 44-jobs on the same machine, which would give a makespan of 108.

When the job of length 50 is added, the makespan of the ideal schedule cannot be less than $A_1+B_1+2C_1(= 44+16+6 = 66)$, as shown above. The following schedule shows that this makespan is possible: $[50, 16]; [44, 16, 3, 3]; [44, 16, 3, 3]; [44, 16, 1, 1, 1, 1]$. Therefore adding the 50-job to a machine that has a 44-job already (and hence a makespan of 64) will give a makespan of 114 and a competitive ratio of $\frac{114}{66}(= 1.727272)$. If the scheduler avoids this by placing it on the fourth machine, then each machine has a load of at least 64, so the final job forces a makespan of $64+88(= 152)$. The competitive divisor is 88, as shown:$[88]; [44, 44]; [50, 16, 16, 3, 1, 1]; [44, 16, 16, 3, 3, 3, 1, 1]$. Therefore, a competitive ratio greater or equal to $\frac{152}{88}(= 1.727272)$ is forced against any scheduling algorithm.

Assume S_A, R, and A_1 as before. To create a type-2 layer, first a B_1 is selected, and all of the B_i jobs will be identical. B_1 must be chosen such that all B_i jobs must be forced onto separate machines to avoid an immediate ratio of $1 + V$. Since the C-row has already been scheduled across all m machines, we have already calculated a divisor D_C such that all jobs through row C can fit on m machines with makespan on any machine less than or equal to D_C. Therefore, the divisor

$$D_B = D_C + B_1.$$

Then B_1 must be sufficiently large that

$$2B_1 + S_C \geq (1 + V)D_B = (1 + V)(D_C + B_1).$$

Hence

$$(1 - V)B_1 \geq (1 + V)D_C - S_C,$$

and so

$$B_1 \geq \frac{(1 + V)D_C - S_C}{(1 - V)}.$$

B_1 is chosen to make the equality true. A_1 must be chosen so that

$$A_m + S_A \geq (1 + V)(A_1 + B_1 + 2C_1).$$

In a type-2 layer,

$$A_1 + 2C_1 \geq A_m,$$

and the divisor when A_m is added is $A_1 + B_1 + 2C_1$. Also,

$$S_A = A_1 + B_1 + S_C.$$

Therefore,

$$A_m + S_A \geq (1 + V)(A_1 + B_1 + 2C_1),$$

and hence

$$2A_1 + 2C_1 + B_1 + S_C \geq A_m + S_A \geq (1 + V)(A_1 + B_1 + 2C_1),$$

from which it follows that

$$(1 - V)A_1 \geq VB_1 + 2VC_1 - S_C$$

and

$$A_1 \geq \frac{(VB_1 + 2VC_1 - S_C)}{1 - V}.$$

A_1 is chosen to make the equality true in the above relation. Note that, in the first step above, this forces

$$A_m = A_1 + 2C_1.$$

Later, this restriction will be relaxed in some cases. However, this restriction gives us the smallest possible A_1, and so the smallest possible R_A, and the largest possible V. At each step, the ratio R_A is reduced. The reduction occurs asymptotically. The limit has no simple closed-form result, but experiment shows that, for V less than or equal to 0.73742, the limit can be reached so that a final type-2 layer can force the competitive ratio of $1 + V$.

More formally, consider the asymptotic behavior of the sequence. The highest possible V for which a solution can be reached is one for which the ratio of $\frac{S_A}{2A_1}$, or $\frac{1}{2R}$, approaches V, as the number of layers increases without bound. This can be found by assuming that

$$\frac{S_A}{2A_1} = V$$

and that the layers are identical in form. (Of course, this condition can never be reached in practice. However, an infinite series of type-2 layers will approach it asymptotically.) Therefore, we assume a steady state, at the end of the C-layer, with $C = \frac{1}{2}$ and $S_C = V$. By construction,

$$\begin{aligned} B_1 &= \frac{(1 + V)D_C - S_C}{1 - V} \\ &= \frac{(1 + V)D_C - V}{1 - V}, \end{aligned}$$

and

$$A_1 = \frac{VB_1 + 2VC_1 - S_C}{1 - V}$$
$$= \frac{VB_1}{1 - V}.$$

By the hypothesis that the asymptotic conditions have been reached, we can assume that

$$D_C = D_A \frac{C_1}{A_1} = (A_1 + B_1 + 2C_1)\frac{1}{2A_1}$$
$$= \frac{A_1 + B_1 + 1}{2A_1}.$$

Finally, we have the relation that

$$\frac{R_A}{S_A} = \frac{1}{2V}$$

or

$$2VA_1 = S_A = A_1 + B_1 + S_C$$
$$= A_1 + B_1 + V.$$

Therefore, we have four equations in the variables A_1, B_1, D_C, and V. The solution was found numerically and is

$$A_1 \cong 6.20899278265808,$$
$$B_1 \cong 2.21087597923733,$$
$$D_C \cong 0.758566573648259,$$
$$V \cong 0.737421563000747.$$

This is the highest value for V that we can achieve with type-1 and type-2 layers. The existing literature shows that, for two and three machines, type-2 layers are *not* needed to achieve optimal values of V. However, for four machines, we need type-2 layers, and these are sufficient to prove the results of this paper. We take up the case of more layers in a forthcoming paper.

3. Four machines. Here we use type-1 and type-2 layers, and they are interspersed. Alternate layers are of each type.

We will prove the following result.

THEOREM 3.1. *For any $\epsilon > 0$, there exists a finite series of jobs that can force any scheduler to a ratio of $\sqrt{3} - \epsilon$ when $m = 4$.*

A construction method will be presented, creating a sequence of jobs for $1 + V = \sqrt{3} - \epsilon$. It will be shown that the sequence terminates, and jobs in each layer (each layer contains four jobs) must be scheduled on the four different machines to avoid a ratio greater than or equal to $1 + V$. After all these layers of jobs are scheduled, a final job will force a ratio of at least $1 + V$.

Note. The type-1 layers are designated by B_i and the type-2 layers by A. These alternate. The sets of jobs are shown below.

B_n	B_n	B_n	B_n
A_n	A_n	A_n	A_n
B_{n-1}	B_{n-1}	B_{n-1}	B_{n-1}
A_{n-1}	A_{n-1}	A_{n-1}	$A_{n-1}+2A_n$
B_{n-2}	B_{n-2}	B_{n-2}	B_{n-2}
A_{n-2}	A_{n-2}	A_{n-2}	$A_{n-2}+2A_{n-1}$
.	.	.	.
.	.	.	.
B_i	B_i	B_i	B_i
A_i	A_i	A_i	A_i+2A_{i+1}
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
B_0	B_0	B_0	B_0
A_0	A_0	A_0	A_0+2A_1
$2A_0$			

S_i and R_i have the usual definition of S_A and R_A for the layer that begins with A_i. Since type-2 layers are being used, each B-row will have four identical jobs, and each A-row will have three identical jobs A_i and one job of size $A_i + 2A_{i+1}$.

3.1. Construction. If $1+V \le 1+\frac{\sqrt{2}}{2}$, then a simple solution using type-1 layers [5] is already known. For $1+\frac{\sqrt{2}}{2} < 1+V < \sqrt{3}$, the form of the sequence of jobs will be as shown in the table above.

We also define

$$S_i = \sum_{j=i}^{n} (A_j + B_j)$$

and

$$R_i = \frac{A_i}{S_i}.$$

Thus S_i is the minimum load on a machine when the jobs n through i have been loaded, as long as no two jobs (of four jobs) from any layer are scheduled on the same machine. Also let

$$V = \sqrt{3} - 1 - \epsilon,$$
$$M = \frac{3V-2}{2}.$$

To begin, define S_0 to be an arbitrary number. Then

$$A_0 = \frac{S_0}{2V},$$
$$R_0 = \frac{A_0}{S_0} = \frac{1}{2V},$$

$$\begin{aligned} B_0 &= S_0 - A_0 - MA_0, \\ S_i &= MA_{i-1}, \qquad i \geq 1, \\ Ai &= \frac{A_{i-1} - 2B_{i-1}}{4}, \qquad i \geq 1, \\ R_i &= \max\left[1, \frac{A_i}{S_i}\right], \qquad i \geq 1. \end{aligned}$$

If $R_i < V$, then

$$\begin{aligned} B_i &= S_i - A_i - MA_i \\ &= S_i - A_i - S_{i+1}, \end{aligned}$$

and we go to index $(i+1)$. If $V \leq R_i \leq 1$, then

$$B_i = S_i - A_i,$$

and the sequence terminates. If $R_i = 1$, then $B_i = 0$, and we redefine $A_i = S_i$, and the sequence terminates. The terminating i becomes n. Note that, by definition of B_n,

$$S_n = A_n + B_n.$$

Also, by definition of B_i,

$$S_i = A_i + B_i + S_{i+1}.$$

Therefore, by induction,

$$S_i = \sum_{j=i}^{n} (A_j + B_j).$$

This represents the smallest possible load on the smallest machine after the jobs n through i have been scheduled if each set of four jobs in a layer has been split among all four machines.

Claim. The procedure described above terminates in finite time for $(1+V) < \sqrt{3}$ and produces a sequence of jobs that will force a ratio of at least $(1+V)$.

LEMMA 3.2. *The procedure described above terminates in finite time for* $(1+V) < \sqrt{3}$.

Proof. Let $(1+V) = \sqrt{3} - \epsilon$. Then $V = \sqrt{3} - 1 - \epsilon$. Since $(1+V) > 1 + \frac{\sqrt{2}}{2}$, we know that

$$\begin{aligned} \epsilon &= \sqrt{3} - (1+V) \\ &< \sqrt{3} - \left(1 + \frac{\sqrt{2}}{2}\right) \\ &< 1.73206 - 1.70710 \\ &= 0.02496 \\ &< \frac{1}{40}. \end{aligned}$$

For $1 \leq i < n$, we have

$$\begin{aligned} R_{i+1} &= \frac{A_{i+1}}{S_{i+1}} \\ &= \frac{\frac{1}{4}A_i - \frac{1}{2}B_i}{MA_i} \\ &= \frac{1}{4M}\frac{A_i - 2(S_i - A_i - S_{i+1})}{A_i} \\ &= \frac{1}{4M}\frac{A_i(1 - (\frac{2}{R_i}) + 2 + 2M)}{A_i} \\ &= \frac{3 + 2M - (\frac{2}{R_i})}{4M} \\ &= \frac{3}{4M} + \frac{1}{2} - \frac{1}{2MR_i}. \end{aligned}$$

We define

$$\delta_i = R_i - R_{i-1},$$

and, therefore,

$$R_i = R_0 + \sum_{j=1}^{i} \delta_j.$$

Recalling that

$$R_0 = \frac{1}{2V},$$

we get

$$\begin{aligned} \delta_1 &= \frac{3}{4M} + \frac{1}{2} - \frac{1}{2MR_0} - \frac{1}{2V} \\ &= \frac{3V + 2MV - 4V^2 - 2M}{4MV} \\ &= \frac{3V + (3V-2)V - 4V^2 - (3V-2)}{4MV} \\ &= \frac{2 - 2V - V^2}{4MV}. \end{aligned}$$

Substituting

$$V = \sqrt{3} - 1 - \epsilon,$$

we get

$$\begin{aligned} \delta_1 &= \frac{2 - 2(\sqrt{3} - 1 - \epsilon) - (\sqrt{3} - 1 - \epsilon)^2}{4MV} \\ &= \frac{2\sqrt{3}\epsilon - \epsilon^2}{4MV} \\ &= \frac{2\sqrt{3}\epsilon - \epsilon^2}{(6V-4)V} \end{aligned}$$

$$
\begin{aligned}
&> \frac{2\sqrt{3}\epsilon - \epsilon^2}{(6\sqrt{3}-10)(\sqrt{3}-1)} \\
&> \frac{2\sqrt{3}\epsilon - \frac{\epsilon}{40}}{28-16\sqrt{3}} \\
&= \frac{(7+4\sqrt{3})(2\sqrt{3}-\frac{1}{40})\epsilon}{(7+4\sqrt{3})(28-16\sqrt{3})} \\
&> \frac{(14\sqrt{3}+24-\frac{7}{40}-\frac{\sqrt{3}}{10})\epsilon}{196-192} \\
&> \left(6+\frac{7}{2}\sqrt{3}-\frac{7}{160}-\frac{\sqrt{3}}{40}\right)\epsilon \\
&> (6+6.062-0.0438-0.0434)\epsilon \\
&> 11\epsilon > 0.
\end{aligned}
$$

Therefore,

$$\delta_1 = R_1 - R_0 > 0,$$

and so $R_1 > R_0$.

For $i \geq 2$,

$$
\begin{aligned}
\delta_i &= \frac{3}{4M} + \frac{1}{2} - \frac{1}{2MR_{i-1}} - \frac{3}{4M} - \frac{1}{2} + \frac{1}{2MR_{i-2}} \\
&= \frac{R_{i-1} - R_{i-2}}{2MR_{i-1}R_{i-2}} \\
&= \frac{\delta_{i-1}}{2MR_{i-1}R_{i-2}}.
\end{aligned}
$$

Since $\delta_1 > 0$, by induction it follows that $\delta_i > 0$, and hence $R_i - R_{i-1} > 0$. Now, if $R_j \geq V$, the sequence stops at j. Otherwise,

$$
\begin{aligned}
\delta_i &= \frac{\delta_{i-1}}{2MR_{i-1}R_{i-2}} \\
&> \frac{\delta_{i-1}}{(3V-2)V^2} \\
&= \frac{\delta_{i-1}}{(3(\sqrt{3}-1)-2)(\sqrt{3}-1)^2} \\
&= \frac{\delta_{i-1}}{(3\sqrt{3}-5)(4-2\sqrt{3})} \\
&= \frac{\delta_{i-1}}{22\sqrt{3}-38} \\
&> 9.5\delta_{i-1} \\
&> (9.5)^{i-1}\delta_1.
\end{aligned}
$$

Therefore,

$$\begin{aligned} R_i &= R_0 + \sum_{j=1}^{i} \delta_j \\ &> R_0 + \delta_1 \sum_{j=1}^{i} (9.5)^{j-1} \\ &= R_0 + \delta_1 \frac{(9.5)^i - 1}{9.5 - 1} \\ &> R_0 + \frac{11\epsilon}{8.5}[(9.5)^i - 1]. \end{aligned}$$

Therefore, R_i increases at an exponential rate, and the sequence will terminate in no more than $c(-\log_{9.5} \epsilon)$ steps, creating a sequence of no more than $[8+c(-8\log_{9.5} \epsilon)]$ jobs for some fixed positive constant c. This proves the lemma.

Remark 1. Note that if $1 + V > \sqrt{3}$ (and therefore ϵ is negative), we have

$$\begin{aligned} \delta_1 &= \frac{2 - 2(\sqrt{3} - 1 - \epsilon) - (\sqrt{3} - 1 - \epsilon)^2}{4MV} \\ &= \frac{2\epsilon\sqrt{3} - \epsilon^2}{4MV} < 0. \end{aligned}$$

Also,

$$\delta_i < (9.5)\delta_{i-1},$$

and so R_i does not increase with i. Hence the method does not terminate for $1+V > \sqrt{3}$. For $1 + V = \sqrt{3}$ (which implies that $\epsilon = 0$), $\delta_i = \delta_1 = 0$, and the method does not terminate. Therefore, this method cannot be used for $1 + V \geq \sqrt{3}$.

LEMMA 3.3. *The jobs from steps n through i can be sorted into four sets totaling no more than $\frac{3}{2}A_i$ and into three sets of no more than $2A_i$.*

Proof. The proof is by induction, beginning with $i = n$. With $i = n$, there are eight jobs

B_n	B_n	B_n	B_n
A_n	A_n	A_n	A_n

with $A_n \geq VS_n$ and $B_n \leq (1 - V)S_n$. Since $V \geq \frac{2}{3}$, $B_n \leq \frac{1}{2}A_n$, and hence

$$A_n + B_n \leq \frac{3}{2}A_n.$$

There are obviously four such sets. Also,

$$A_n + B_n + B_n \leq A_n + A_n.$$

Therefore, the three sets $\{A_n, A_n\}$,$\{A_n, B_n, B_n\}$,$\{A_n, B_n, B_n\}$ satisfy the first condition. Therefore, the lemma is true for $i = n$.

Now, assume that it is true for the jobs in sets n through i. Then those jobs can be collected into three sets that total to a number less than or equal to $2A_i$. These three sets, which are interchangeable for purposes of this proof, can be labeled X_{3i}.

Similarly, we have four sets X_{4i}, each of which total to a number less than or equal to $\frac{3}{2}A_i$.

Now, consider the three sets $\{A_{i-1}, A_{i-1}\}$,$\{A_{i-1}, B_{i-1}, B_{i-1}, X_{3i}, X_{3i}\}$, and $\{A_{i-1}+2A_i, B_{i-1}, B_{i-1}, X_{3i}\}$.

The total of the jobs in the first set is $2A_{i-1}$. Since the total in X_{3i} is less than or equal to $2A_i$, by the induction hypothesis, the second set's total is less than or equal to the third set's total. Therefore, we need only consider the third set. Please note that since

$$A_i = \frac{(A_{i-1} - 2B_i)}{4},$$

by construction, we have

$$A_{i-1} = 2B_{i-1} + 4A_i.$$

The total load of the third machine is

$$\begin{aligned} A_{i-1} + 2A_i + B_{i-1} + B_{i-1} + X_{3i} &\\ &\leq A_{i-1} + 2B_{i-1} + 4A_i \\ &\leq \ A_{i-1} + A_{i-1}. \end{aligned}$$

Therefore, all three machines have loads that are less than or equal to $2A_{i-1}$.

Now consider the four sets $\{A_{i-1}, B_{i-1}, X_{3i}\}$,$\{A_{i-1}, B_{i-1}, X_{3i}\}$,$\{A_{i-1}, B_{i-1}, X_{3i}\}$, and $\{A_{i-1} + 2A_i, B_{i-1}\}$.

Since the total load in set X_{3i} is less than or equal to $2A_i$, each set has a total load less than or equal to

$$\begin{aligned} A_{i-1} + B_{i-1} + 2A_i &\\ &= A_{i-1} + \frac{1}{2}A_{i-1} = \frac{3}{2}A_{i-1}. \end{aligned}$$

So Lemma 3.3 is true for $i-1$. Therefore, by induction, it is true for all $0 \leq i \leq n$.

LEMMA 3.4. *The jobs $\{B_i, B_i, B_i, B_i\}$ must each be scheduled on a separate machine to prevent having a ratio $1+V$ or more. Furthermore, the four A-level jobs must each be scheduled on a separate machine to prevent having a ratio $1+V$ or more.*

Proof. Clearly it is true for B_n, since that situation is simply scheduling four identical jobs. If two of them are scheduled on one machine, the ratio is $2 > 1+V$. Since

$$R_n = \frac{A_n}{S_n} \geq V,$$

we have $A_n \geq VS_n$. Scheduling two jobs of size A_n on one machine gives a load of

$$\begin{aligned} (B_n + A_n) + A_n &\geq S_n + VS_n \\ &= (1+V)S_n. \end{aligned}$$

The divisor is S_n, so this forces a ratio greater than or equal to $1+V$. Therefore, the lemma is true for n. Assume that it is true for the jobs in sets n through i. Then all jobs in sets n through i must be scheduled evenly. In this case, the smallest

possible load on any machine is S_i. If two jobs of size B_{i-1} are scheduled on one machine, its load is greater than or equal to

$$\begin{aligned} S_i + 2B_{i-1} &= S_i + A_{i-1} - 4A_i \\ &= MA_{i-1} + A_{i-1} - 4A_i \\ &= \frac{M+1}{M} S_i - 4A_i \\ &= \frac{M+1}{MR_i} A_i - 4A_i \\ &= \frac{M+1-4MR_i}{MR_i} A_i. \end{aligned}$$

By Lemma 3.3, the divisor must be less than or equal to

$$\begin{aligned} B_{i-1} + X_{4i} &\leq \frac{1}{2} A_{i-1} - 2A_i + \frac{3}{2} A_i \\ &= \frac{1}{2}(A_{i-1} - A_i) \\ &= \frac{1}{2}\left(\frac{S_i}{M} - A_i\right) \\ &= \frac{1}{2}\left(\frac{A_i}{R_i M} - A_i\right) \\ &= \frac{1-R_iM}{2R_iM} A_i. \end{aligned}$$

Therefore, the ratio must be greater than or equal to

$$\begin{aligned} &\frac{2(M+1-4R_iM)}{1-R_iM} \\ &= \frac{2M+2-8R_iM}{1-R_iM} \\ &= 2 + \frac{2M-6R_iM}{1-R_iM}. \end{aligned}$$

Since $M \cong 0.09807 < 0.1$ and $R_i < V < \sqrt{3} - 1 < 0.733$ for $i < n$,

$$\begin{aligned} &2 + \frac{2M-6R_iM}{1-R_iM} \\ &= 2 - \frac{M(6R_i-2)}{1-R_iM} \\ &> 2 - \frac{0.1(4.398-2)}{0.9267} \\ &= 2 - 0.258 \\ &= 1.742 > (1+V). \end{aligned}$$

Therefore, two B_i cannot be scheduled on the same machine without exceeding the competitive ratio $1+V$. So if the lemma is true for n through i, then it is true for B_i.

The jobs $\{A_{i-1}, A_{i-1}, A_{i-1}, A_{i-1} + 2A_i\}$ must each be scheduled on separate machines to prevent having a ratio of $1+V$ or more.

As the fourth job is scheduled, any attempt to put it on the same machine as one of the other three will result in a machine whose load is at least

$$N_{i-1} = S_i + B_{i-1} + 2A_{i-1} + 2A_i = MA_{i-1} + 2A_{i-1} + \frac{1}{2}A_{i-1}$$
$$= A_{i-1}\left(M + \frac{5}{2}\right).$$

By Lemma 3.3, the denominator is no more than

$$D_{i-1} = \frac{3}{2}A_{i-1}.$$

So the ratio is at least

$$\frac{2M+5}{3} = \frac{3V-2+5}{3}$$
$$= \frac{3V+3}{3}$$
$$= 1+V.$$

As the second and third jobs are scheduled, any attempt to put two jobs on the same machine will result in a machine load of at least

$$S_i + B_{i-1} + 2A_{i-1} = A_{i-1}\left(M + \frac{5}{2} - 2\frac{A_i}{A_{i-1}}\right) = N_{i-1} - 2\frac{A_i}{A_{i-1}}.$$

The denominator is

$$A_{i-1} + B_{i-1} = D_{i-1} - 2\frac{A_i}{A_{i-1}}.$$

Therefore, this ratio is greater than $\frac{N_{i-1}}{D_{i-1}}$, previously shown to be greater than or equal to $1+V$. Therefore, by induction, Lemma 3.4 is true.

LEMMA 3.5. *The final job of $2A_0$ will force a ratio of at least $1+V$.*

Proof. By the previous lemma, if a scheduler schedules any two jobs from the same layer on one machine, it immediately forces a ratio greater than $1+V$. Therefore, the scheduler is forced to put one from each layer on each of the four machines. So the smallest possible load on any machine is the sum of the smallest load in each layer. This sum is S_0. By construction,

$$S_0 = 2A_0V.$$

Therefore, the numerator when a job of $2A_0$ is added will be at least

$$2A_0V + 2A_0 = (1+V)2A_0.$$

By Lemma 3.3, the denominator can be no more than $2A_0$. Therefore, the ratio is at least $1+V$. Therefore, this finite sequence forces a ratio of

$$1+V = \sqrt{3} - 1 - \epsilon$$

against any scheduler strategy.

Example 2. Let $\epsilon = 10^{-8}$. Then $1+V = 1.732050798$, $M = .098076196$, and the sequence in Table 3.1 is generated.

TABLE 3.1
Sequence of variables to generate layers for $1 + V = \sqrt{3} - 10^{-8}$.

i	S_i	R_i	A_i	B_i
0	7,320,507,976	0.683012711	5,000,000,000	1,830,126,994
1	490,380,982	0.683012832	334,936,503	122,595,180
2	32,849,298	0.68301415	22,436,536	8,212,273
3	2,200,490	0.683028555	1,502,998	550,084
4	147,408	0.683185975	100,707	36,824
5	9,877	0.684905818	6,765	2,449
6	663	0.70364391	467	151
7	46	0.901864135	41	5

TABLE 3.2
Job sequence to force $1 + V = \sqrt{3} - 10^{-8}$.

5	5	5	5
41	41	41	41
151	151	151	151
467	467	467	549
2,449	2,449	2,449	2,449
6,765	6,765	6,765	7,699
36,824	36,824	36,824	36,824
100,707	100,707	100,707	114,237
550,084	550,084	550,084	550,084
1,502,998	1,502,998	1,502,998	1,704,412
8,212,273	8,212,273	8,212,273	8,212,273
22,436,536	22,436,536	22,436,536	25,442,532
122,595,180	122,595,180	122,595,180	122,595,180
334,936,503	334,936,503	334,936,503	379,809,575
1,830,126,994	1,830,126,994	1,830,126,994	1,830,126,994
5,000,000,000	5,000,000,000	5,000,000,000	5,669,873,006
10,000,000,000			

Since $R_7 > V$, the sequence terminates at $n = 7$. Therefore, the series of jobs in Table 3.2 guarantees a ratio greater than or equal to $\sqrt{3} - 10^{-8}$, against any scheduler strategy.

In a forthcoming paper, we discuss how this approach can lead to better lower bounds for a larger number of machines.

REFERENCES

[1] S. ALBERS, *Better bounds for on-line scheduling*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 130–139.

[2] Y. BARTAL, A. FIAT, H. KARLOFF, AND R. VOHRA, *New algorithms for an ancient scheduling problem*, J. Comput. System Sci., 51 (1995), pp. 359–366.

[3] Y. BARTOL, H. KARLOFF, AND Y. RABANI, *A better lower bound for on-line scheduling*, Inform. Process. Lett., 50 (1994), pp. 113–116.

[4] B. CHEN, A. VAN VLIET, AND G. J. WOEGINGER, *New lower and upper bounds for on-line scheduling*, Oper. Res. Lett., 16 (1994), pp. 221–230.

[5] U. FAIGLE, W. KERN, AND G. TURAN, *On the performance of on-line algorithms for partition problems*, Acta Cybernet., 9 (1989), pp. 107–119.

[6] A. FIAT AND G. J. WOEGINGER, EDS., *Online Algorithms*, Springer-Verlag, Berlin, 1998.

[7] G. GALAMBOS AND G. J. WOEGINGER, *An on-line scheduling heuristic with better worst-case ratio than Graham's list scheduling*, SIAM J. Comput., 22 (1993), pp. 349–355.

[8] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 1563–1582.
[9] R. L. Graham, *Bounds on multiprocessor timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.
[10] D. R. Karger, S. J. Phillips, and E. Torng, *A better algorithm for an ancient scheduling problem*, J. Algorithms, 20 (1996), pp. 400–430.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 736–753

NEW LOWER BOUND TECHNIQUES FOR DYNAMIC PARTIAL SUMS AND RELATED PROBLEMS*

THORE HUSFELDT† AND THEIS RAUHE‡

Abstract. We study the complexity of the dynamic partial sum problem in the cell-probe model. We give the model access to nondeterministic queries and prove that the problem remains hard. We give the model access to the right answer ± 1 as an oracle and prove that the problem remains hard. This suggests which kind of information is hard to maintain.

From these results, we derive a number of lower bounds for dynamic algorithms and data structures: We prove lower bounds for dynamic algorithms for existential range queries, reachability in directed graphs, planarity testing, planar point location, incremental parsing, and fundamental data structure problems like maintaining the majority of the prefixes of a string of bits. We prove a lower bound for reachability in grid graphs in terms of the graph's width. We characterize the complexity of maintaining the value of any symmetric function on the prefixes of a bit string.

Key words. cell-probe model, partial sum, dynamic algorithm, data structure

AMS subject classifications. 68Q17, 68Q10, 68Q05, 68P05

PII. S0097539701391592

1. Introduction.

The partial sum problem is to maintain n bits $x_1, \ldots, x_n \in \{0, 1\}$ that are subject to updates

$$\textit{update}(i)\text{: change } x_i \text{ to } 1 - x_i$$

and compute queries about the partial sums $x_1 + \cdots + x_i$.

It is easy to construct data structures that provide either very fast updates (by computing the answer from scratch after each query) or very fast queries (by recomputing all partial sums after each update). However, in many partial sum problems—and in many dynamic problems in general—we cannot have both. This trade-off between *update time* and *query time* was established by Fredman and Saks [15], who showed that, with the parity query

$$\textit{parity}(i)\text{: return } x_1 + \cdots + x_i \mod 2,$$

the partial sum problem requires time $\Omega(\log n / \log\log n)$ per operation on the unit-cost RAM with logarithmic cell size. In other words, even the least significant bits of the partial sums are hard to maintain.

The motivation for the present paper is that the hardness of the problem depends on the following query: If the parity query is replaced by

$$\textit{or}(i)\text{: return "yes" iff } x_1 + \cdots + x_i \geq 1 \text{ (equivalently, return } x_1 \vee \cdots \vee x_i),$$

*Received by the editors June 29, 2001; accepted for publication (in revised form) October 22, 2002; published electronically April 23, 2003. A preliminary version of this paper appeared in *Proceedings of the 25th ICALP*, Lecture Notes in Comput. Sci. 1443, Springer-Verlag, Berlin, 1998, pp. 67–78.

http://www.siam.org/journals/sicomp/32-3/39159.html

†Department of Computer Science, Lund University, P.O. Box 118, SE–221 00 Lund, Sweden (thore@cs.lu.se). This author was partially supported by Swedish TFR, the ESPRIT Long Term Research Programme of the EU, project 20244 (ALCOM-IT), and BRICS (Basic Research in Computer Science), a center of the Danish National Research Foundation.

‡IT University of Copenhagen, Glentevej 67, DK–2400 Copenhagen NV, Denmark (theis@it-c.dk). This author was partially supported by the ESPRIT Long Term Research Programme of the EU, project 20244 (ALCOM-IT), and BRICS (Basic Research in Computer Science), a center of the Danish National Research Foundation.

then a Van Emde Boas tree provides an implementation in time $O(\log \log n)$ per operation, which is exponentially faster.

We show which queries are hard in this sense.

General partial sum queries. Consider two other natural partial sum queries:

$$\begin{aligned} &\textit{majority}(i)\text{: return 1 iff } x_1 + \cdots + x_i \geq \lceil \tfrac{1}{2} i \rceil, \\ &\textit{equality}(i)\text{: return 1 iff } x_1 + \cdots + x_i = \lceil \tfrac{1}{2} i \rceil. \end{aligned}$$

We can formulate these problems as database queries like "Did as many male as female guests arrive before noon?" or "Are more French than English talks scheduled between Tuesday and Friday?" Similarly, these problems can be viewed as natural *range query* problems in computational geometry.

Proposition 3 of the present paper shows that both problems require time $\Omega(\log n / \log \log n)$ per operation, just as parity. We then extend our analysis of the majority problem to the class of *threshold* functions and characterize the complexity of the resulting partial sum problem in terms of the size of the threshold in Proposition 4. This connects the majority problem, where the threshold is $\frac{1}{2}i$, and the *or* problem above, where the threshold is 1. Finally, we generalize this to the entire class of *symmetric* functions in Proposition 11.

Intriguingly, the resulting bounds closely resemble the corresponding results from Boolean circuit complexity, where these problems have been studied intensively, hinting at a connection between the dynamic and parallel realms.

Main contribution. Our main technical and conceptual contributions are lower bounds for partial sum problems in very strong models of computation. All our other results follow from these bounds.

The idea is to provide the query algorithm with well-defined parts of the answer for free without reducing the problem's complexity. We phrase the results for the *signed partial sum problem*. The problem is to maintain a string $x \in \{-1, 0, +1\}^n$ under the following operations:

$$\begin{aligned} &\textit{update}(i, a)\text{: change } x_i \text{ to } a \in \{-1, 0, +1\}, \\ &\textit{query}(i)\text{: return } x_1 + \cdots + x_i \mod 2. \end{aligned}$$

We prove two theorems about this problem.

Theorem 1 shows that, even in models with *nondeterministic* queries (defined and discussed in section 2), the partial sum problem requires time $\Omega(\log n / \log \log n)$ per operation. It is known that this is also the deterministic complexity of the problem [9, 15], so nondeterminism does not help.

Theorem 3 studies the same problem in a *promise* setting, where the (deterministic) query algorithm receives an almost correct answer for free. The updates are as before, and the query is

$$\textit{parity}(i, s)\text{: return } x_1 + \cdots + x_i \mod 2 \text{ provided that } \left| s - \sum_{j=1}^{i} x_j \right| \leq 1$$

(otherwise, the behavior of the query algorithm is undefined).

We show that this problem still requires $\Omega(\log n / \log \log n)$ per operation.

Lower bounds for dynamic algorithms. We present some applications to dynamic algorithms and data structure problems other than partial sums. Because Theorems 1 and 3 hold in very strong models of computation, we can construct powerful reductions.

We can show that the existential problem for orthogonal range queries in the plane requires time $\Omega(\log^{1/2} n)$ per operation (Proposition 2). We also present bounds for planar point location in monotone subdivisions [5, 26], reachability in upward planar digraphs [28], and incremental parsing of balanced parentheses [11]. We show that these problems require time $\Omega(\log n/\log\log n)$ per operation (Propositions 5–8). It is known [10, 14, 17, 23] that this is also a lower bound for reachability in grid graphs. However, grid graphs of constant width allow a reachability algorithm in time $O(\log\log n)$ per operation [4], an exponential improvement. We prove a lower bound that is parameterized by the width w of the graph: Proposition 10 states that dynamic reachability for grid graphs of width $w = O(\log n/\log\log n)$ requires time $\Omega(w)$ per operation, bridging the gap between the two results.

Apart from the bound for the existential range query problem, for which the authors recently proved a stronger bound using a different technique [3], all these bounds are new and the best known.

Related work. Fredman introduced the partial sum problem as a "toy problem which is both tractable and surprisingly interesting" [13], and it has been the focal point of many investigations of dynamic complexity in a variety of models [15, 31]. We reason within the cell-probe model of Fredman [12] and Yao [30] with some extensions to cope with our stronger modes of computation. The model can be viewed as a nonuniform version of the random access computer with arbitrary register instructions. Lower bounds are especially valid on RAMs with unit-cost instructions and logarithmic cell size. The success of this model is partly due to the validity of these bounds in light of schemes like hashing, indirect addressing, bucketing, pointer manipulation, or recent algorithms that exploit the parallelism inherent in unit-cost instructions. For these reasons, the cell-probe model has arguably become the model of choice for lower bounds for dynamic computation.

Theorems 1 and 3 are proved by extending the chronogram method, which was introduced by Fredman and Saks [15] and got its name in [7].

The prefix parity problem was solved in [15], but no nontrivial lower bounds for the majority or equality problems follow from that. The results from [6, 21, 22, 29] can be seen to imply $\Omega(\log\log n/\log\log\log n)$ lower bounds using an entirely different technique based on Ajtai's result [2]; and [19] reports $\Omega((\log n/\log\log n)^{1/2})$ for equality and $\Omega(\log n/(\log\log n)^2)$ for the majority.

2. Nondeterminism in dynamic algorithms.

2.1. Example: Range queries. We can illustrate our concept of nondeterministic queries using the *existential range query* problem. The object is to maintain a set $S \subseteq \{1,\dots,n\}^2$ of points in the plane under the following operations:

update(x): add $x \in \{1,\dots,n\}^2$ to S, or remove it if it is already there,

exists(y): return "yes" iff S contains a point x in the rectangle defined by the origin and y, i.e., such that $x_1 \le y_1$ and $x_2 \le y_2$.

With *nondeterministic* queries, the problem is very easy: guess a point and verify that it is in $S \cap R$. This shows that positive instances of this problem have short,

maintainable witnesses—the points themselves. On the other hand, it is known that, for deterministic computation, this problem requires time $\Omega(\log n/\log\log n)$ [3]; we prove a somewhat weaker bound in Proposition 2. Thus the hardness of this problem lies in maintaining precisely the kind of information that nondeterminism provides for free.

However, this is not true for all problems; we shall show that queries about the *size* $|R \cap S|$ remain hard even with nondeterminism. Thus we see that the hardness of the two problems, both of which have the same deterministic complexity, hinges on information of a fundamentally different kind.

Another example from computational geometry is *dynamic convex hull*, the problem of maintaining the convex hull of a set of points S, where points are inserted and removed. The query operation asks whether the query point q lies inside or outside the convex hull of S. Again, we can solve this problem with a trivial update algorithm that simply stores S in a large table. (In the cell-probe model, we do not worry about memory space; otherwise, we can use standard dictionaries.) The nondeterministic query guesses three points from S and verifies that the query point lies in the triangle spanned by these points—a well known result in plane geometry asserts that this is necessary and sufficient.

Thus we have identified a class of dynamic problems, namely, those with fast nondeterministic queries. Problems in this class have positive instances with short witnesses, and these witnesses can be maintained by an efficient data structure. This encompasses the class of problems where the outcome of each query depends on only a small number of updates. Contrast this with the problems identified in [15], where each update affects only a small number of queries, e.g., dictionary problems.

2.2. A model for nondeterministic query algorithms. We introduce a model for nondeterministic query algorithms for dynamic *decision* problems, where the query returns 0 or 1. We allow query algorithms to nondeterministically load a value into a memory cell. The semantics are as usual: The value returned by a nondeterministic query is 1 unless all nondeterministic choices return 0. For example, the following program solves in constant time the existential range query problem, storing all points from S in a two-dimensional array M:

update(x_1, x_2):
  $M[x_1, x_2] := 1 - M[x_1, x_2]$,

exists(y_1, y_2):
  guess $x_1 \le y_1$ and $x_2 \le y_2$
  return $M[x_1, x_2]$.

We should mention that we have not defined the *side-effects* of a nondeterministic query algorithm, i.e., the effect of its assignments to memory. This can be done in a number of ways; for example, we might say that if there are computations (i.e., sequences of nondeterministic choices) that result in "1," the algorithm will execute one of these computations; otherwise, it will execute a computation leading to "0." Our lower bound is immune to precisely how these effects are defined, since the hard operation sequence constructed in the proof needs only a single query, which happens at the very end.

2.3. Signed partial sum. The *signed partial sum* problem is to maintain a string of letters $x \in \{-1, 0, +1\}^n$, initially 0^n, under updates that change the letters

of x and queries about the parity of the prefix sums of x:

$$\begin{aligned} &\textit{update}(i,a)\text{: change } x_i \text{ to } a \in \{-1,0,+1\},\\ &\textit{query}(i)\text{: return } x_1 + \cdots + x_i \mod 2. \end{aligned}$$

The data structure of Dietz [9] solves this problem deterministically in time $O(\log n/\log\log n)$ per operation with logarithmic cell size. The next theorem states that nondeterministic queries can do no better. We state this theorem as a trade-off between update and query time.

THEOREM 1. *Every nondeterministic algorithm for the signed partial sum problem with cell size b, update time t_u, and query time t_q must satisfy*

$$t_q = \Omega\left(\frac{\log n}{\log(bt_u \log n)}\right). \tag{1}$$

Also, the lower bound holds even if the algorithm requires

$$0 \le x_1 + \cdots + x_i \le \left\lceil \frac{\log n}{\log(bt_u \log n)} \right\rceil \tag{2}$$

for all i after each update.

The proof is given in the next section.

Note that the query cannot distinguish $+1$ from -1 (since $1 = -1 \mod 2$), so a data structure for the signed partial sum problem structure can treat -1 as $+1$. The reason for introducing -1 in the problem is the balancing condition (2), which continues previous work [19] on extending the chronogram method.

In section 5.2, we state a further generalization of Theorem 1, relating the terms in (1) and (2).

2.4. Lower bound for existential range queries. We give a lower bound of size $\Omega(\log^{1/2} n)$ for the existential range query problem; we consider cell size $b = \log n$ for concreteness. The value of this result lies in its simplicity; it provides a good illustration of how to apply Theorem 1. Using a different technique [3], the authors with Alstrup have since established $\Omega(\log n/\log(bt_u))$, which is optimal. However, before the present paper, no lower bound better than $\Omega(\log\log n/\log\log\log n)$ was known for this problem (which is rather central—see the discussion by Agarwal [1]), so the result provides an exponential yet, by now, outdated improvement.

Following [3], we start with the *existential marked ancestor problem*. Consider a full rooted tree with nodes V, number of leaves n, height h, and arity d, where

$$h = \log^{1/2} n, \quad d = 2^h. \tag{3}$$

Let $\pi(v)$ denote the nodes on the path from v to the root (including v). The problem is to maintain a subset of *marked* nodes $M \subseteq V$ under the following operations:

mark(v): insert $v \in V$ in M,

unmark(v): remove $v \in V$ from M,

exists(v): return "yes" iff any of v's ancestors are marked, i.e., if $\pi(v) \cap M \neq \emptyset$.

The *counting marked ancestors* problem supports the same updates, and the query is

parity(v): return $|M \cap \pi(v)| \mod 2$, the parity of the number of marked ancestors of v.

The parity prefix sum problem is a special case of this problem, where the tree is a path. We begin by showing that the problem is hard also for d-ary trees, where $d = \log^{1/2} n$.

LEMMA 1. *Every nondeterministic algorithm for counting marked ancestors in trees with update time t_u requires query time*

$$t_q = \Omega\left(\frac{\log n}{\log^{1/2} n + \log(t_u \log n)}\right).$$

Proof. Let x be a length n instance to the signed partial sum problem. We assume that $\log^{1/2} n$ is an integer. Consider a data structure for the counting marked ancestor problem for a tree T with parameters as in (3), and update and query time t_u and t_q. The ith leaf v_i of T corresponds to x_i. We will maintain that the parity of the number of marked ancestors to v_i is the parity of the ith prefix sum in x, i.e.,

$$|\pi(v_i) \cap M| = x_1 + \cdots + x_i \pmod 2.$$

Thus the time for a partial sum query is the same as the time for a marked ancestor query, t_q. To maintain the invariant whenever x_i is changed (and thus the parity of all prefix sums $\geq i$ are changed), we change the marking of the root of a number of disjoint subtrees in T, whose leaves correspond to $x_i, \ldots, x_n$. These roots are the right siblings of $\pi(v_{i-1})$, so there are at most dh updates. Thus the update time is at most $t_u dh = O(2^{\log^{1/2} n} t_u \log^{1/2} n)$. Now Theorem 1 implies the bound on the query time. □

The proof of the next proposition contains the crucial application of nondeterminism to transform a counting problem into an existential one.

PROPOSITION 1. *Existential Marked Ancestor requires time $\Omega(\log^{1/2} n)$ per operation.*

Proof. Consider an algorithm for the existential problem with update time t_u and query time t_q, and let T be an instance of the counting marked ancestor problem. Construct 2^h new instances T_w indexed by bit strings $w \in \{0,1\}^h$. We maintain that the markings in the first instance $T_{00\ldots0}$ are the same as in T. In general, the ith bit of w is cleared iff the markings in T_w on level i are the same as in T. More precisely, if v is a node on level i, we have

$$(v \text{ marked in } T_w) = w_i \oplus (v \text{ marked in } T),$$

where $\oplus$ denotes exclusive or. The crucial observation is the following: Let v be a leaf. Then w is the characteristic vector of $\pi(v) \cap M$ iff the path $\pi(v)$ in T_w is unmarked.

Whenever a node in T is marked or unmarked, we must update all 2^h instances, so the update time is $2^h t_u$. For a query, we guess the characteristic vector of $\pi(v) \cap M$ and verify that $\pi(v)$ is unmarked in T_w. This takes time $t_q + O(1)$. We finish the proof by applying the above lemma. □

Finally, we present the application to range queries.

PROPOSITION 2. *Existential Range Query requires time $\Omega(\log^{1/2} n)$ per operation.*

Proof. Embed the tree from the marked ancestor problem in the first quadrant of the plane, with the root in the origin and the nodes at depth i spread out evenly on the diagonal $y = -x + d^h - d^{h-i}$. The query rectangle has its upper-right corner in the queried node. □

2.5. Discussion. Analyzing the above proof, we see that the algorithm used in the reduction actually solves the complement of the problem; we use it to verify $\pi(v) \cap M = \emptyset$. Thus the proof also yields a bound on the nondeterministic complexity of the *emptiness* problem (to return "yes" iff the query rectangle is empty). In other words, there is no short, maintainable witness to the absence of points in the plane.

In contrast, the emptiness problem in one dimension does admit a fast nondeterministic algorithm, since we can maintain a doubly linked list of the inserted points, and the query can guess both the immediate predecessor and immediate successor of a query interval and verify that they are neighbors in S. Using a Van Emde Boas tree, this can be implemented in time $O(\log \log n)$ per update and constant query time.

3. Proof of Theorem 1. We consider a specific sequence of operations that consists of a number of updates followed by a single query. The update sequence is chosen at random from a set U defined in section 3.5.

3.1. Model of computation. The computational model is an extension of the cell-probe model [12, 30]; since there is only a single query in the hard sequence of operations constructed in our proof, which happens at the very end of the sequence, we can model query algorithms by nondeterministic decision trees.

More precisely, a *cell-probe* algorithm consists of a family of trees, one for each operation, and a memory $M \in \{0, \dots, 2^b - 1\}^*$. We refer to the elements of M as *cells*, each of which can store a b-bit number. To each update we associate a decision-assignment tree as in [15]. There are two types of nodes: *Read* nodes are 2^b-ary and labeled by a memory address, and computation proceeds to the child identified at that address; *write* nodes are unary and labeled by a memory address and a b-bit value, with the obvious semantics.

To each query we associate a nondeterministic decision tree of arity 2^b whose internal nodes are labeled by a memory address or by "$\exists$." The leaves are labeled 0 or 1 to represent the possible answers to the query. We define the value $qM \in \{0, 1\}$ computed by a query tree q on memory M to be 1 if there exists a path from the root to a leaf with label 1. A witness of such an accepting computation is the description of the choices for the $\exists$ nodes. We let q_i denote the query tree corresponding to $query(i)$. The query time t_q is the height of the largest query tree, and the update time t_u is the height of the largest update tree. We account only for memory reads and writes and for nondeterministic choices; all other computation is for free.

3.2. Updates and epochs. Each update sequence in U is described by a binary string $u \in \{0, 1\}^*$. Each bit represents an update $update(j, a)$. The parameters for these updates will be specified in section 3.5. The update sequences $u \in U$ are split into d substrings each corresponding to an *epoch*. It turns out to be convenient that time flows backward, so epoch 1 corresponds to the end of u. In general, the update string is an element in $U = U_d U_{d-1} \cdots U_1$, where $U_t = \{0, 1\}^{e(t)}$ and where $e(t)$ is the length of epoch t such that $e(t) + \cdots + e(1) = \lfloor n^{t/d}/d \rfloor$. The length of the entire update sequence is $\lfloor n/d \rfloor$. The size of d and hence the growth rate of $e(t)$ are given by

$$d = \left\lceil \frac{\log n}{\log(b t_u \log n)} \right\rceil. \tag{4}$$

The goal is to establish that $t_q \in \Omega(d)$.

3.3. Time stamps and nondeterminism. To each cell we associate a time stamp when it is written. A cell receives time stamp t if some update during epoch t writes to it, and none of the subsequent updates during epochs $t-1$ to 1 write to it.

For an update sequence $u \in U$, let M^u denote the memory resulting from these updates (recall that updates are restricted to perform deterministically), starting with some arbitrary initial contents corresponding to the initial instance 0^n.

For index i and update string u, let $T(i,u)$ denote the set of time stamps that are found on every accepting computation path of q_i on M^u. If there are no accepting computations, the set is empty. More formally, let w denote a witness for a computation path of q_i on M^u, and let $A(i,u)$ denote the set of witnesses that leads to accepting computations of q_i on M^u. Let for a moment $T(i,u,w)$ denote the set of time stamps encountered by the computation of q_i on M^u that is witnessed by w. Then $T(i,u) = \bigcap \{\, T(i,u,w) \mid w \in A(i,u) \,\}$ if $A(i,u) \neq \emptyset$, and $T(i,u) = \emptyset$ otherwise.

The simple lemma below is the tool to identify a read of a cell with time stamp t by nondeterministic queries.

LEMMA 2. *If M^u and M^v differ only on cells with time stamp t, then $q_i M^u \neq q_i M^v$ implies $t \in T(i,u) \cup T(i,v)$.*

Proof. Suppose, on the contrary, that $q_i M^u \neq q_i M^v$ and $t \notin T(i,u) \cup T(i,v)$. Assume without loss of generality that $q_i M^u = 1$ and $q_i M^v = 0$. Since $t \notin T(i,u)$ and $q_i M^u = 1$, there is an accepting computation path that avoids cells with time stamp t. However, this computation might as well be executed on M^v, by the premise. Hence q_i has an accepting computation on M^v as well, contradicting $q_i M^v = 0$. □

3.4. Lower bound on query time. The update sequences are chosen such that, even if two sequences differ only in a single epoch, they still result in very different instances. To each update sequence $u \in U$ we associate the query vector $q^u = (q_1 M^u, q_2 M^u, \ldots, q_n M^u) \in \{0,1\}^n$. Update sequences that differ only in epoch t are called *t-different*.

LEMMA 3. *No Hamming ball of diameter $\frac{1}{8}n$ can contain more than $|U_t|^{9/10}$ query vectors from t-different update sequences for large n.*

The difficult part is constructing a set of update sequences for which the statement is true, which we present in section 3.5. The proof itself is as in [15] and is provided in section 3.5 for completeness.

Write $U_{>t}$ for $U_d \cdots U_{t+1}$, the set of update sequences prior to epoch t, and write $U_{<t}$ for $U_{t-1} \cdots U_1$, the set of update sequences in epoch t to epoch 1. Assume for the rest of this section that $t_{\mathrm{q}} = \mathrm{O}(\log n)$; else there is nothing to prove. The worst-case query time t_{q} is at least the average of $|T(i,u)|$ over choices of $i \in \{1,\ldots,n\}$ and $u \in U$, so

$$|U| n t_{\mathrm{q}} \geq \sum_{u \in U} \sum_{i=1}^{n} |T(i,u)| = \sum_{t=1}^{d} \sum_{u \in U_{>t}} \sum_{w \in U_{<t}} \sum_{v \in U_t} \sum_{i=1}^{n} \big(t \in T(i,uvw)\big).$$

The next lemma tells us how many $v \in U_t$ fail to make the last sum exceed $\frac{1}{16}n$.

LEMMA 4. *Fix any epoch $1 \leq t \leq d$ and past and future updates $x \in U_{<t}$, $y \in U_{>t}$. For large n, at least half of the update sequences $u \in xU_t y$ satisfy $\big|\{\, 1 \leq i \leq n \mid t \in T(i,u) \,\}\big| \geq \frac{1}{16}n$ if $t_{\mathrm{q}} = \mathrm{O}(\log n)$.*

Proof. Consider the set $V \subseteq xU_t y$ of updates after which fewer than $\frac{1}{16}n$ queries encounter time stamp t; i.e., xuy for $u \in U_t$ is in V if

$$\big|\{\, 1 \leq i \leq n \mid t \in T(i,xuy) \,\}\big| < \tfrac{1}{16}n.$$

We will bound the size of V below $\frac{1}{2}|U_t|$.

To this end, partition V into equivalence classes such that u and v are in the same class iff M^u and M^v agree on all cells except maybe those with time stamp t. We first bound the number of such classes. Since all cells with time stamp greater than t have identical content (they depend only on the common prefix x), we need only to analyze the amount of information distributed among cells with time stamps $t-1$ to 1. The number of cells written during the last $t-1$ epochs is at most $r = t_{\mathrm{u}} \cdot (e(t-1) + \cdots + e(1))$. Note that at most $n2^{t_{\mathrm{q}} b}$ different cells appear in the entire forest of query trees. The number of different ways we can choose such r cells and fix their content to some value in $\{0, \ldots, 2^b - 1\}$ is bounded by

$$(n2^{t_{\mathrm{q}} b} \cdot 2^b)^r \leq |U_t|^{o(1)}, \tag{5}$$

where the inequality uses (4). That is, $|U_t|^{o(1)}$ bounds the number of equivalence classes of V.

It remains to bound the size of each class. Consider two query vectors q^u and q^v for u and v in the same equivalence class. Then

$$|q^u - q^v| \leq \tfrac{1}{8}n \tag{6}$$

because $\frac{15}{16}n$ entries of each vector depend only on cells with time stamps other than t. On these cells, the memories are indistinguishable and therefore yield the same result by Lemma 2. By (6), all vectors from the same class end up in a Hamming ball of diameter $\frac{1}{8}n$, so Lemma 3 tells us that there can be only $|U_t|^{\frac{9}{10}}$ of them. We conclude that the size of V is bounded by $|U_t|^{\frac{9}{10}} \cdot |U_t|^{o(1)}$, which is less than $\frac{1}{2}|U_t|$ for large n. □

By this lemma we obtain for large n

$$|U| n t_{\mathrm{q}} \geq \sum_{t=1}^{d} |U_{>t}| \cdot |U_{<t}| \cdot \tfrac{1}{16}n \cdot \tfrac{1}{2}|U_t| = \tfrac{1}{32}nd|U|$$

and hence $t_{\mathrm{q}} \geq \frac{1}{32}d$ as desired.

3.5. Update scheme. The technical part that remains is to exhibit a set of update sequences U satisfying Lemma 3. There are a number of ways to do this; the following construction is one which simultaneously anticipates our needs in section 6 and satisfies the balancing condition (2).

To alleviate notation, we assume that n/d is an integer. Consider the updates in epoch t, and index them as $u_1 \cdots u_{e(t)} \in U_t$. If $u_i = 0$, then nothing happens in the ith update. Else it performs $\mathit{update}(j, a)$, where the update position j is given below. The new value is $a = (-1)^r$, where $r = 1 + u_1 + \cdots + u_i \mod 2$, so the nonzero updates in u alternate between -1 and $+1$, starting with $+1$. The position of the affected letter is defined as follows. Write x as a table of dimension $d \times n/d$ like this:

$$\begin{bmatrix} x_1 & x_{d+1} & & x_{n-d+1} \\ \vdots & \vdots & \cdots & \vdots \\ x_d & x_{2d} & & x_n \end{bmatrix}.$$

All updates in epoch t will affect only the letters in row t. The updates of an epoch are spread out evenly from left to right across that row, so the distance between two

of them is

$$\left\lfloor \frac{n/d}{e(t)} \right\rfloor. \tag{7}$$

In summary, the ith update in epoch t affects the letter in row t and the column given by $(i-1)\cdot\lfloor (n/d)/e(t)\rfloor + 1$.

This update scheme satisfies the statement in Lemma 3.

Proof of Lemma 3. Let xU_ty be any set of t-different update sequences. Pick any $u \in U_t$, and consider any Hamming ball of diameter $\frac{1}{8}n$ that contains query vector q^{xuy}. We will bound the number of $v \in U^t$ with query vector q^{xuy} ending up in that Hamming ball.

Let $w \in U_t$ record the difference between u and v; i.e., the ith letter of w is 1 iff u and v differ on the ith letter. Now let w' denote the string of prefix sum parities of w, i.e.,

$$w'_i = w_1 + \cdots + w_i \mod 2, \quad 1 \le i \le e(t).$$

It is easy to see that w' records the difference between the query vectors resulting from u and v. Indeed, each 1 in w' yields an interval of indices where the vectors differ, and the length of this interval is d times the distance given by (7). In other words, each 1 in w' contributes as many points to the Hamming distance between the resulting query vectors. So, if we let $|w'|_1$ denote the number of 1's in w', the Hamming distance between two query vectors is at least

$$|w'|_1 \cdot d \cdot \left\lfloor \frac{n/d}{e(t)} \right\rfloor \ge \tfrac{1}{2}|w'|_1 \cdot \frac{n}{e(t)}, \tag{8}$$

where we have used that $\lfloor a \rfloor \ge \frac{1}{2}a$ for $a \ge 1$.

By the triangle inequality, the maximum Hamming distance between two query vectors in the same ball is $\frac{1}{8}n$. This bounds the number of 1's in w' to $\frac{1}{4}e(t)$ for large n. Hence the number of choices for w' is bounded by

$$\sum_{i=0}^{\frac{1}{4}e(t)} \binom{e(t)}{i} < 2^{\frac{9}{10}e(t)} \tag{9}$$

for large n. This also bounds the number choices of $v \in U_t$ since there is a one-to-one correspondence between v and w'. ∎

The prefix sums of instances resulting from our scheme are small: Let x denote an instance resulting from our scheme from the initial instance 0^n. Let x^t denote the string resulting from only the updates in epoch t, and write x as $x^1 + \cdots + x^d$; this works because no two epochs write in the same positions. Then

$$\sum_{j=1}^{i} x_j = \sum_{j=1}^{i}\sum_{t=1}^{d} x_j^t = \sum_{t=1}^{d}\sum_{j=1}^{i} x_j^t \in \{0, \ldots, d\}$$

because the prefix sum of every x^t is 0 or 1 by construction. It can be checked that the balancing bound (2) holds at all times.

Another important feature of this update scheme, which we will use to prove Theorem 3, is that, if x and y result from t-different updates, then $x^r = y^r$ for $r \ne t$

and hence

$$\left| \sum_{j=1}^{i} x_j - \sum_{j=1}^{i} y_j \right| \le 1 \tag{10}$$

for all i.

4. Partial sum queries. The next result shows that the majority and equality problems defined in the introduction are just as hard as the parity query from [15]. The proof is a simple application of Theorem 1.

PROPOSITION 3. *The prefix equality and prefix majority problems satisfy*

$$t_{\mathrm{q}} = \Omega\left(\frac{\log n}{\log(t_{\mathrm{u}} b \log n)} \right).$$

Proof. We first give the proof for prefix equality. Let $d = \lceil \log n / \log(b t_{\mathrm{u}} \log n) \rceil$.

An instance $x \in \{-1, 0, +1\}^n$ of signed partial sum is encoded as the binary string x' by

$$-1 \mapsto 00, \quad 0 \mapsto 01, \quad +1 \mapsto 11.$$

We maintain $d+1$ strings $y^{(0)}, \dots, y^{(d)}$ as

$$y^{(t)} = (00)^t (01)^{d-t} x'.$$

Let $t_{\mathrm{u}} = t_{\mathrm{u}}(n)$ denote the update time of our prefix equality algorithm. Whenever x is changed, we make at most $2d+2$ updates in the strings $y^{(t)}$; so the update time is $(2d+2) \cdot t_{\mathrm{u}}(n + 2d + 2)$.

Index the strings $y^{(t)}$ from $-2d$ to $2n-1$. We then have

$$\sum_{j=-2d}^{2i-1} y_j^{(t)} = d - t + i + \sum_{j=1}^{i} x_j, \qquad 0 \le t \le d, \quad 1 \le i \le n. \tag{11}$$

Hence, in order to find the ith prefix sum of x, our algorithm can nondeterministically guess the sum $s \in \{0, \dots, d\}$; we can assume from the balancing condition (2) in Theorem 1 that the sum is in that set and verify $y_{-2d}^{(s)} + \cdots + y_{2i-1}^{(s)} = d + i$, which is the case iff *equality*$(2d+2i)$ on $y^{(s)}$ returns 1. The conclusion is by Theorem 1.

The same bound must hold for the majority problem since we can write

$$\begin{aligned} x_1 + \cdots + x_i = \left\lceil \tfrac{1}{2} i \right\rceil \text{ iff } & x_1 + \cdots + x_i \\ & \ge \left\lceil \tfrac{1}{2} i \right\rceil \wedge \bar{x}_1 + \cdots + \bar{x}_i \ge \left\lceil \tfrac{1}{2} i \right\rceil, \end{aligned}$$

where $\bar{x}_i = 1 - x_i$, and these negated values are easily maintained. □

To study this kind of problem in a general, let the *threshold* ϑ be an integer function such that $\vartheta(i) \in \{0, \dots, \lceil \frac{1}{2} i \rceil\}$. The query in the *prefix threshold problem for* ϑ is

threshold(i): return "yes" iff $x_1 + \cdots + x_i \ge \vartheta(i)$.

Prefix majority is the special case $\vartheta(i) = \lceil \frac{1}{2} i \rceil$; prefix-or is $\vartheta(i) = 1$. Now, for our lower bound, our assumption on ϑ is that there are integers $p(1) < p(2) < \cdots < p(i) < \cdots$ such that $\vartheta(p(i)) = i$. We call such functions *nice* for lack of a better word. It is

reasonable to assume that ϑ is monotonically increasing; the niceness assumption also prevents it from skipping points.

PROPOSITION 4. *Let $t_u = t_u(n)$ and $t_q = t_q(n)$ denote the update and query time of any cell size b implementation of the prefix threshold problem for a nice threshold ϑ. Then $t_q = \Omega(\log \vartheta / \log(t_u b \log \vartheta))$.*

Proof. The proof is not difficult but is tedious. The idea is to stretch an instance for a threshold problem, padding it with sufficiently many 0's or 1's to turn it into a majority problem.

Let ϑ be a nice function, and let $p(1), \ldots, p(n)$ be such that $\vartheta(p(i)) = i$. Assume we have an algorithm for the prefix problem for ϑ with the parameters given in the statement of the theorem. We will construct an algorithm for the majority with instance $x \in \{0,1\}^n$. Construct a bit string y as

$$y = 0 \cdots 0 x_1 x_1 0 \cdots 0 x_2 x_2 0 \cdots 0 x_n x_n,$$

where the letters of x are at positions $p(1)-1, p(1), p(2)-1, p(2), \ldots, p(n)-1, p(n)$; denote the length of y by $m = p(n)$.

The string y can be maintained in time $2t_u(m)$ for each update of x. For the query, note that $2x_1 + \cdots + 2x_i = y_1 + \cdots + y_{p(i)}$, so

$$x_1 + \cdots + x_i \geq \left\lceil \tfrac{1}{2} i \right\rceil \quad \text{iff} \quad y_1 + \cdots + y_{p(i)} \geq i = \vartheta(p(i)),$$

so the majority function (left-hand side) can be expressed in terms of the threshold function ϑ (right-hand side). Hence the query time is $t_q(m)$. However, from the bound on the complexity of the majority function, we know that

$$t_q(m) = \Omega\left(\frac{\log n}{\log(t_u(m) b(m) \log n)} \right).$$

The stated bound follows by substituting $\vartheta(m)$ for n. □

To gauge the strength of this result, we mention that the problem can be solved on the unit-cost RAM with logarithmic cell size in time $O((\log \vartheta / \log\log n) + \log\log n)$ per update (if $\vartheta(1), \ldots, \vartheta(n)$ can be computed in the preprocessing stage of the algorithm). The left term in the expression stems from a search tree, and the right term stems from a priority queue, which vanishes for cell size $b = \Omega(\log^2 n)$; details are omitted. A comparison with Proposition 4 shows that the lower bound is tight for logarithmic cell size and $\vartheta = \Omega(\log^{\log\log n} n)$. For smaller thresholds, the bounds leave a gap of size $O(\log\log n)$.

We consider a more general class of query functions in section 6.2.

5. Applications to dynamic algorithms. Theorem 1 suggests a new approach for proving lower bounds for dynamic algorithms by employing nondeterminism in the reduction from signed partial sum. We demonstrate this with a number of examples in this section. The results are presented for cell size $b = \log n$ for concreteness. Some of the reductions extend previous work of the authors with Søren Skyum [19].

5.1. Nested brackets. Consider the problem of maintaining a nested structure, i.e., a string x with round and square brackets under the following operations:

change(i, a): change x_i to a, where a is a round or square opening or a closing bracket or whitespace.

balance: return "yes" iff the brackets in x are properly nested.

This problem was studied in [11], where an algorithm with polylogarithmic update time is presented.

PROPOSITION 5. *Maintaining a string of nested brackets requires time* $\Omega(\log n/\log\log n)$ *per operation.*

Proof. Consider a deterministic algorithm for this problem, and consider an instance $x \in \{0,-1,+1\}^n$ to signed partial sum. Let b_i be an encoding of x_i given by

$$+1 \mapsto))\sqcup, \quad 0 \mapsto)\sqcup\sqcup, \quad -1 \mapsto \sqcup\sqcup\sqcup,$$

where "$\sqcup$" stands for space. Let c be the string "$\sqcup($" consisting of a single space and an opening bracket. We maintain a balanced string of brackets uvw, where $u = c^{2n}$, $v = b_1 b_2 \dots b_n$, and $w =)^{n-s}\sqcup^s$, where $s = x_1 + \cdots + x_n$. It is easy to see that uvw balances and can be maintained by a constant number of updates per update in x. For any prefix size i, this construction enables efficient verification of a nondeterministic guess g of the prefix sum $x_1 + \cdots + x_i$: Place a closing square bracket on the last $\sqcup$ of b_i and an opening square bracket on the $\sqcup$ of the first c of suffix c^{i+g} of u. This modification keeps uvw balanced iff g is the right guess of prefix sum $x_1 + \cdots + x_i$. The conclusion is by Theorem 1. □

5.2. Dynamic graph algorithms. Our techniques improve the lower bounds of a number of well-studied graph problems considered in [19].

Tamassia and Preparata [28] present an algorithm for the class of *upward planar source-sink graphs* that runs in time $O(\log n)$ per operation. These digraphs have a planar embedding where all edges point upward (meaning that their projection on some fixed direction is positive) and where exactly one node has indegree 0 (the source) and exactly one node has outdegree 0 (the sink). The updates are

insert(u,v): insert an edge from u to v,
delete(u,v): delete the edge from u to v if it exists,
reachable(u,v): return "yes" iff there is a path from u to v.

The updates have to preserve the topology of the graph, including the embedding.

PROPOSITION 6. *Dynamic reachability in upward planar source-sink graphs requires time* $\Omega(\log n/\log\log n)$ *per operation.*

Planarity testing is to maintain a planar graph where the query asks whether a new edge violates the planarity of the graph. Italiano, Poutré, and Rauch [20] present an efficient algorithm for a version of this problem, and a strong lower bound is exhibited by Fredman and Henzinger [14]. Our lower bound also holds for *upward* planarity testing, where the topology is further restricted to upward planar graphs. The updates insert and delete edges as above, and the query is as follows:

planar(u,v): return "yes" iff the graph remains upward planar after insertion of edge (u,v).

This problem was studied by Tamassia [27], who found an $O(\log n)$ upper bound.

PROPOSITION 7. *Upward planarity testing requires time* $\Omega(\log n/\log\log n)$ *per operation.*

A classical problem in computational geometry is *planar point location*: given a subdivision of the plane, i.e., a partition into polygonal regions induced by the straight-line embedding of a planar graph, determine the region of query point $q \in \mathbf{R}^2$.

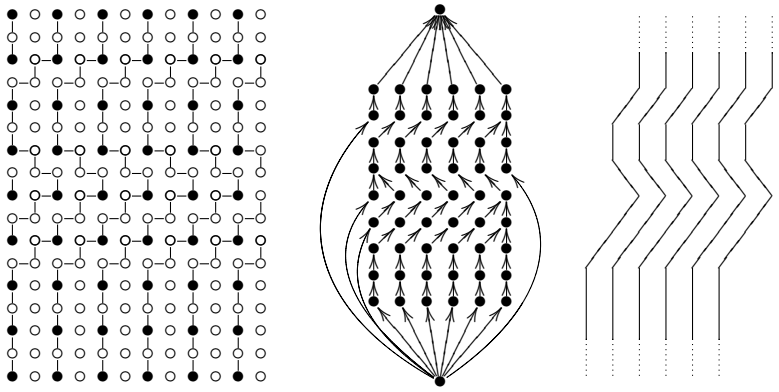


FIG. 1. *Planar graphs corresponding to* $x = (0, 0, +1, +1, -1, 0, +1, 0)$. *Left: Grid graph. Even grid points are marked* •*; odd grid points are marked* ○. *Middle: Upward planar source-sink graph. Right: Monotone planar subdivision.*

An important restriction of the problem considers only *monotone* subdivisions, where the subdivision consists of polygons that are monotone (so no horizontal line crosses any polygon more than twice). In the dynamic version of this problem, updates manipulate the geometry of the subdivision. Preparata and Tamassia [26] give an algorithm that runs in time $O(\log^2 n)$ per operation; this was improved to query time $O(\log n)$ by Baumgarten, Jung, and Mehlhorn [5]. The lower bound for this problem in [19] applies only to algorithms returning the name of the region containing the queried point. The techniques of the present paper extend this bound to work for simpler decision queries like

query(x): return "yes" iff x is in the same polygon as the origin.

PROPOSITION 8. *Planar point location requires time* $\Omega(\log n / \log\log n)$ *per operation, even in monotone subdivisions.*

Traditionally, lower bounds in computational geometry are proved in an algebraic, comparison-based model (see [25] for a textbook account) that is broken by standard RAM operations like indirect addressing, bucketing, hashing, etc. Cell-probe lower bounds for that field are lacking.

To explain our reduction, we turn to the conceptually very simple class of *grid graphs*. The vertices of a grid graph of width w and height h are integer points (i, j) in the plane for $1 \le i \le w$ and $1 \le j \le h$. All edges have length 1 and are parallel to the axes. The dynamic reachability problem for these graphs is the following:

flip(x, y): add an edge between $x \in [w] \times [h]$ and $y \in [w] \times [h]$ or remove it if it exists,

reachable(x, y): return "yes" iff there is a path from x to y.

There are several well-known constructions that prove a lower bound for this problem [10, 14, 17, 23], but our proof translates to the other problems in Propositions 6–8. The details in these constructions are omitted; Figure 1 illustrates the structures arising in the reductions.

PROPOSITION 9. *Dynamic reachability in grid graphs requires time* $\Omega(\log n/\log\log n)$ *per operation.*

Proof. From an instance $x \in \{0, \pm 1\}^n$ to signed partial sum, we build a grid graph on the points $\{0, \dots, 2w\} \times \{0, \dots, 2n\}$, where $w = \lceil \log n / \log\log n \rceil$. We will exploit the balancing constraint (2) of Theorem 1 to keep the instance within this width.

For every i and j, consider any point with even coordinates $(2i, 2j-2)$, drawn as $\bullet$ in Figure 1, and connect it to one of the three even grid points above it using  ,  , or  , depending on whether $x_j = +1$, 0, or -1, respectively. The idea is that the path from $(0, 0)$ mimics the prefix sums of x in that it passes through $(2s, 2j)$ iff $x_1 + \cdots + x_j$ equals s. Hence a guess of the sum can be verified by a single reachability query in the graph.

It remains to note that the graph can be maintained efficiently. Any changed letter in x incurs $O(w)$ edges to be inserted or deleted. So, if the update time of the graph algorithm is polylogarithmic, then the graph can be maintained in polylogarithmic time. The bound follows from Theorem 1. □

The width of the hard graph above is logarithmic in the height, while the graphs constructed in [10, 14, 17, 23] are square. Hence narrow grid graphs are as hard as square ones. However, this is not true for *very* narrow graphs: It is known that the reachability problem for grid graphs of *constant* width can be solved in time $O(\log\log n)$ by [4], an exponential improvement. This leaves open the question of what happens for graphs of sublogarithmic width. To answer this, we introduce a subtler statement of Theorem 1.

THEOREM 2. *Let* $d = O(\log n / \log(bt_u \log n))$ *be an integer function. Every nondeterministic algorithm for signed partial sum with cell size* b*, update time* t_u*, and query time* t_q *must satisfy* $t_q = \Omega(d)$*. The lower bound holds even if the algorithm requires* $0 \le x_1 + \cdots + x_i \le d$ *for all* i *after each update.*

This result implies a lower bound for grid graphs that smoothly connects the two extremes between linear and constant width. A similar parameterization can be done for all our problems.

PROPOSITION 10. *For every* $w = O(\log n / \log\log n)$*, dynamic reachability in grid graphs of width* w *requires time* $\Omega(w)$ *per operation.*

6. Refinement. We now take a somewhat subtler approach to our basic question than we take in section 2. Instead of nondeterminism, we study the performance of query algorithms in a *promise* setting. We assume that the query algorithm for signed partial sum receives a value s that is promised to be *close to* (but not known to be equal to) the right sum and then decides between right and wrong values.

The *partial sum refinement* problem can be phrased as follows: Maintain a string $x \in \{0, \pm 1\}^n$, initially 0^n, under the following operations:

update(i, a): change x_i to $a \in \{-1, 0, +1\}$,

parity(i, s): return $x_1 + \cdots + x_i \mod 2$ provided that $|s - \sum_{j=1}^{i} x_j| \le 1$ (for other values of s, the behavior of the query algorithm is undefined).

The problem gets its name from the following alternative definition, where the query operation is replaced by

refine(i, s): return 1 if $s = \sum_{j=1}^{i} x_j$ and 0 if $s \ne \sum_{j=1}^{i} x_j$, provided that $|s - \sum_{j=1}^{i} x_j| \le 1$. (For other values of s, the behavior of the query algorithm is undefined.)

The two problems are computationally equivalent.

THEOREM 3. *Let d be an integer function such that $d = \mathrm{O}(\log n/\log(t_{\mathrm{u}} b \log n))$. Every algorithm for partial sum refinement with cell size b, update time t_{u}, and query time t_{q} must satisfy $t_{\mathrm{q}} = \Omega(d)$. Moreover, this is true even for algorithms that require $0 \le x_1 + \cdots + x_i \le d$ for all i after each update.*

6.1. Proof of Theorem 3. Most of the technical work for this result was already done in section 3.5, where we found that the instances resulting from two t-different updates have close prefix sums (10).

The query trees in our computational model are now deterministic decision trees as in [15]. However, there are more of them: we associate a tree q_i^s to each query *parity*(i, s), yielding $n(2n+1)$ trees. (We could reduce this number to $n(d+1)$ by the balancing constraint, but that does not improve the bounds.)

For update string u, we write q_i^u for the query tree q_i^s corresponding to the "right guess" $s = x_1 + \cdots + x_i$, where x is the instance resulting from updates u. The *query vector* is $(q_1^u M, \ldots, q_n^u M)$, i.e., the responses yielded by guessing right every time. We let $T(i, u)$ denote the time stamps encountered by q_i^u on M^u and compare this with the construction in section 3.3.

The next lemma corresponds to Lemma 2 and shows that our update scheme constructs different instances whose prefix sums are so close that the query trees cannot use the (almost correct) value given to them.

LEMMA 5. *For t-different update sequence $u, v \in U_t$, if M^u and M^v differ only on cells with time stamp t, then, for all i,*

$$q_i^u M^u \neq q_i^v M^v \quad \textit{implies } t \in T(i,u) \cup T(i,v).$$

Proof. Assume, to the contrary, for some such t, u, v, and i, that $t \notin T(i, v)$ and $q_i^u M^u \neq q_i^v M^v$. Let x and y denote the input instances resulting from u and v, respectively. Let s denote $\sum_{j=1}^{i} x_j$. By (10) and without loss of generality, $\sum_{j=1}^{i} y_j = s+1$. By correctness, $q_i^s M^u = q_i^{s+1} M^u$. Since the computation path for $q_i^{s+1} M^v$ does not encounter time stamp t, this computation might as well be executed on M^u with the same result; i.e., $q_i^{s+1} M^u = q_i^{s+1} M^v = q_i^s M^u = q_i^u M^u$. However, this contradicts our assumption $q_i^u M^u \neq q_i^v M^v = q_i^{s+1} M^v$. □

The rest of the proof can be reused almost ad verbatim.

6.2. The partial sum problem for symmetric functions. Theorem 3 acts as an important ingredient in characterizing the dynamic complexity of all the *symmetric functions*, generalizing the results for the threshold functions of the last section. A Boolean function is *symmetric* if it depends only on the number of 1's in the input $x = (x_1, \ldots, x_n)$. The symmetric functions include some of the most well-studied functions in complexity theory like parity, majority, and the threshold functions.

In general, we can describe every symmetric function f in n variables by its *spectrum*, a string in $\{0,1\}^{n+1}$ whose ith letter is the value of f on inputs where exactly i variables are 1. The *boundary* of a spectrum s is the smallest value ϑ such that $s_{\lfloor \vartheta \rfloor} = s_{\lfloor \vartheta \rfloor + 1} = \cdots = s_{\lfloor n - \vartheta \rfloor}$. For instance, the boundary of the parity or majority functions is $\frac{1}{2}n$, and for the threshold functions with threshold ϑ, the boundary is $\min(\vartheta, n - \vartheta)$.

Let $\langle f_n \rangle = (f_1, \ldots, f_n)$ be a sequence of symmetric Boolean functions where the ith function f_i takes i variables. The *dynamic prefix problem for* $\langle f_n \rangle$ is to maintain

a bit string $x \in \{0,1\}^n$ under the following operations:

$$\begin{aligned} &\textit{update}(i)\text{: change } x_i \text{ to } \neg x_i,\\ &\textit{query}(i)\text{: return } f_i(x_1,\ldots,x_i). \end{aligned}$$

For example, taking f_i to be the parity function on i variables, we have the prefix parity problem of [15], and taking f_i to be the threshold function for $\vartheta(i)$, we have the problem from Proposition 4.

PROPOSITION 11. *Let ϑ be a nice function, and let $\langle f_n\rangle$ be a sequence of symmetric functions where $f_i\colon \{0,1\}^i \to \{0,1\}$ has boundary $\vartheta(i)$. Let t_{u} and t_{q} denote the update and query time of any cell size b implementation of the dynamic prefix problem for $\langle f_n\rangle$. Then $t_{\mathrm{q}} = \Omega(\log\vartheta/\log(t_{\mathrm{u}} b \log\vartheta))$.*

Proof. First assume that f_i's boundary is in the middle, i.e., $\vartheta(i) = \frac{1}{2}i$. Let $x \in \{+1,0,-1\}^n$ denote an instance to prefix refinement, and define d and maintain $d+1$ strings as in the proof for Proposition 3. Using the data structure for $\langle f_n\rangle$, we perform *refine*(i,g) as follows. Let s be the spectrum for f_{2i+2d}. Since its boundary is in the middle, it is the case that

$$s_{d+i-1}s_{d+i}s_{d+i+1} \in \{001, 010, 011, 100, 101, 110\}.$$

We consider only the case 001 above—the other cases are treated similarly. Recall that we can assume $x_1 + \cdots + x_i \in \{g-1, g, g+1\}$. Let r_{-1}, r_0, and r_{+1} denote the answer of *query*$(2d+2i)$ on $y^{(g-1)}$, $y^{(g)}$, and $y^{(g+1)}$, respectively. By (11) in the proof of Proposition 3, if $g = x_1 + \cdots + x_i$, then $r_{-1}r_0r_{+1} = s_{d+i-1}s_{d+i}s_{d+i+1} = 001$. If instead $g-1$ is the correct sum, then $r_{-1}r_0 = 01$, and finally if $g+1$ is the correct sum, then $r_0r_{+1} = 00$. Hence these three cases for g can be distinguished by the above three queries, and they hence determine the correct answer for *refine*(i,g). The bound then follows from Theorem 3.

The rest of the proof is a padding argument that "stretches" the above to work for smaller ϑ similarly to the proof of Proposition 3. We omit the details. □

Intriguingly, the bound in the proposition is precisely the same bound as for the size-depth trade-off for Boolean circuits for these functions [16, 8, 24].

Acknowledgments. The authors thank Arne Anderson, Gerth Stølting Brodal, Faith Fich, Peter Bro Miltersen, and Sven Skyum for valuable comments about various aspects of this work.

REFERENCES

[1] P. K. AGARWAL, *Range searching*, in Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke, eds., CRC Press, Boca Raton, FL, 1997, pp. 575–598.

[2] M. AJTAI, *A lower bound for finding predecessors in Yao's cell probe model*, Combinatorica, 8 (1988), pp. 235–247.

[3] S. ALSTRUP, T. HUSFELDT, AND T. RAUHE, *Marked ancestor problems*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 534–543.

[4] D. A. M. BARRINGTON, C.-J. LU, P. B. MILTERSEN, AND S. SKYUM, *Searching constant width mazes captures the AC^0-hierarchy*, in Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1373, Springer-Verlag, Berlin, 1998, pp. 73–83.

[5] H. BAUMGARTEN, H. JUNG, AND K. MEHLHORN, *Dynamic point location in general subdivisions*, in Proceedings of the 3rd Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1992, pp. 250–258.

[6] P. Beame and F. Fich, *Optimal bounds for the predecessor problem and related problems*, J. Comput. System Sci., 65 (2002), pp. 38–72.
[7] A. M. Ben-Amram and Z. Galil, *Lower bounds for data structure problems on RAMs*, in Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1991, pp. 622–631.
[8] B. Brustmann and I. Wegener, *The complexity of symmetric functions in bounded-depth circuits*, Inform. Process. Lett., 25 (1987), pp. 217–219.
[9] P. F. Dietz, *Optimal algorithms for list indexing and subset rank*, in Proceedings of the 1st Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 382, Springer-Verlag, Berlin, 1989, pp. 39–46.
[10] D. Eppstein, *Dynamic connectivity in digital images*, Inform. Process. Lett., 62 (1997), pp. 121–126.
[11] G. S. Frandsen, T. Husfeldt, P. B. Miltersen, T. Rauhe, and S. Skyum, *Dynamic algorithms for the Dyck languages*, in Proceedings of the 4th Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 955, Springer-Verlag, Berlin, 1995, pp. 98–108.
[12] M. L. Fredman, *Observations on the complexity of generating quasi-Gray codes*, SIAM J. Comput., 7 (1978), pp. 134–146.
[13] M. L. Fredman, *The complexity of maintaining an array and computing its partial sums*, J. ACM, 29 (1982), pp. 250–260.
[14] M. L. Fredman and M. R. Henzinger, *Lower bounds for fully dynamic connectivity problems in graphs*, Algorithmica, 22 (1998), pp. 351–362.
[15] M. L. Fredman and M. E. Saks, *The cell probe complexity of dynamic data structures*, in Proceedings of the 21st ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 345–354.
[16] J. T. Håstad, *Almost optimal lower bounds for small depth circuits*, in Proceedings of the 18th ACM Symposium on Theory of Computing, ACM, New York, 1986, pp. 6–20.
[17] T. Husfeldt, *Fully dynamic transitive closure in plane dags with one source and one sink*, in Proceedings of the 3rd European Symposium on Algorithms, Lecture Notes in Comput. Sci. 955, Springer-Verlag, Berlin, 1995, pp. 199–212.
[18] T. Husfeldt and T. Rauhe, *Hardness results for dynamic problems by extensions of Fredman and Saks' chronogram method*, in Proceedings of the 25th ICALP, Lecture Notes in Comput. Sci. 1443, Springer-Verlag, Berlin, 1998, pp. 67–78.
[19] T. Husfeldt, T. Rauhe, and S. Skyum, *Lower bounds for dynamic transitive closure, planar point location, and parentheses matching*, Nordic J. Comput., 3 (1996), pp. 323–336.
[20] G. F. Italiano, J. A. La Poutré, and M. H. Rauch, *Fully dynamic planarity testing in planar embedded graphs*, in Proceedings of the 1st Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 726, Springer-Verlag, Berlin, 1993, pp. 212–223.
[21] P. B. Miltersen, *Lower bounds for union–split–find related problems on random access machines*, in Proceedings of the 26th ACM Symposium on Theory of Computing, ACM, New York, 1994, pp. 625–634.
[22] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson, *On data structures and asymmetric communication complexity*, in Proceedings of the 27th ACM Symposium on Theory of Computing, ACM, New York, 1995, pp. 103–111.
[23] P. B. Miltersen, S. Subramanian, J. S. Vitter, and R. Tamassia, *Complexity models for incremental computation*, Theoret. Comput. Sci., 130 (1994), pp. 203–236.
[24] S. Moran, *Generalized lower bounds derived from Hastad's main lemma*, Inform. Process. Lett., 25 (1987), pp. 383–388.
[25] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, Berlin, 1985.
[26] F. P. Preparata and R. Tamassia, *Fully dynamic point location in a monotone subdivision*, SIAM J. Comput., 18 (1989), pp. 811–830.
[27] R. Tamassia, *On-line planar graph embedding*, J. Algorithms, 21 (1996), pp. 201–239.
[28] R. Tamassia and F. P. Preparata, *Dynamic maintenance of planar digraphs, with applications*, Algorithmica, 5 (1990), pp. 509–527.
[29] B. Xiao, *New Bounds in Cell Probe Model*, Doctoral dissertation, University of California San Diego, San Diego, CA, 1992.
[30] A. C. Yao, *Should tables be sorted?*, J. ACM, 28 (1981), pp. 615–628.
[31] A. C. Yao, *On the complexity of maintaining partial sums*, SIAM J. Comput., 14 (1985), pp. 277–288.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 754–792

THE RECONSTRUCTION OF DOUBLED GENOMES*

NADIA EL-MABROUK† AND DAVID SANKOFF‡

Abstract. The genome can be modeled as a set of strings (chromosomes) of distinguished elements called genes. Genome duplication is an important source of new gene functions and novel physiological pathways. Originally (ancestrally), a duplicated genome contains two identical copies of each chromosome, but through the genomic rearrangement mutational processes of reciprocal translocation (prefix and/or suffix exchanges between chromosomes) and substring reversals, this simple doubled structure is disrupted. At the time of observation, each of the chromosomes resulting from the accumulation of rearrangements can be decomposed into a succession of conserved segments, such that each segment appears exactly twice in the genome. We present exact algorithms for reconstructing the ancestral doubled genome in linear time, minimizing the number of rearrangement mutations required to derive the observed order of genes along the present-day chromosomes. Somewhat different techniques are required for a translocations-only model, a translocations/reversals model, both of these in the multichromosomal context (eukaryotic nuclear genomes), and a reversals-only model for single chromosome prokaryotic and organellar genomes. We apply these methods to the yeast genome, which is thought to have doubled, and to the liverwort mitochondrial genome, whose duplicate genes are unlikely to have arisen by genome doubling.

Key words. genome duplication, genome rearrangement, signed genes, reversal, translocation, Hannenhalli–Pevzner graph, exact polynomial algorithms

AMS subject classifications. 68Q25, 68Q17, 05C38, 05C62, 05C85

PII. S0097539700377177

1. Introduction.

In almost all the genomes which have been studied, there are some genes that are present in two or more copies. These copies may be identical or may have some differences, and they may be adjacent on a single chromosome or dispersed on different chromosomes throughout the genome. There are a number of different ways in which duplicate genes can arise; perhaps the most spectacular mechanism is the simultaneous doubling of the entire genome. Normally a lethal accident of meiosis or other reproductive step, if genome doubling can be resolved in the organism and eventually fixed as a normalized diploid state in a population, simultaneous doubling constitutes a duplication of the entire genetic complement. It transcends other mechanisms for gene duplication in that not only is one copy of each gene free to evolve its own function (or to lose function, becoming a pseudogene and mutating randomly, eventually beyond recognition), but it can evolve in concert with any subset of the thousands of other extra gene copies (cf. [14] for accounts of gene family coevolution). Whole new physiological pathways may emerge, involving novel functions for many of these genes. Genome duplication is thus a likely source of rapid and far-reaching evolutionary progress. Its rarity does not detract from its

*Received by the editors August 29, 2000; accepted for publication (in revised form) December 13, 2002; published electronically April 23, 2003. This work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada. Various parts of this paper were presented at the RECOMB conference in Lyons in April 1999, the Genome Informatics Workshop (Tokyo) in December 1999, the JOBIM conference in Montpellier in May 2000, and at the DCAF conference in Montreal in September 2000 and appear in preliminary form in the proceedings of these conferences. The present version contains important corrections and improvements.

http://www.siam.org/journals/sicomp/32-3/37717.html

†Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, CP 6128 Succursale Centre-ville, Montréal, QB H3C 3J7, Canada (mabrouk@iro.umontreal.ca).

‡Centre de Recherches Mathématiques, Université de Montréal, CP 6128 Succursale Centre-ville, Montréal, QB H3C 3J7, Canada (sankoff@crm.umontreal.ca).

importance.

For some genomes, recent polyploidy is easily detected due to the presence of a complete set of duplicated chromosomes. However, in most cases, all we can observe are duplicated chromosomal segments scattered throughout the genome.

Evidence for the effects of genome duplication has shown up across the eukaryote spectrum. More than two hundred million years ago, the vertebrate genome may have undergone two duplications [4, 20, 31], though at least one of these remains controversial [38, 21, 25, 8, 13]. Although numerous reversals and reciprocal translocations have subsequently occurred, the number of such chromosome rearrangements has been sufficiently modest that hundreds of conserved paralogous segments can be detected in the human genome since the ancient duplications; similar observations hold for the mouse genome [28, 29] and for less intensively mapped vertebrate genomes. More recent genome duplications are known to have occurred in some vertebrate lines, such as the frogs [40], the salmoniform fish [31], and the zebrafish [33].

Another example is given by the comparison of chromatin-eliminating Ascaridae with other nematodes. This comparison suggests that somatic cells of these worms have discarded a good proportion of the genes present in germ cells, possibly because these are redundant duplicates arising through genomic doubling some 200 million years ago [27].

Genome duplication is particularly prevalent in plants. Comparison of the well-studied rice [1], oats (wild and domestic), corn [1, 15], and wheat [26] genomes indicate several occurrences in the cereal lineage. Soybeans [36], Arabidopsis [24, 3], rapeseed [34], and other cultivars have genome duplications in their ancestry. Paterson et al. have presented convincing evidence that one or more genome duplications also occurred much earlier in plant evolution [32].

Following the complete sequencing of all Saccharomyces cerevisiae chromosomes, the prevalence of gene duplication has led to the hypothesis that this yeast genome is also the product of an ancient doubling [35, 39].

What of bacteria and other prokaryotes? In 1985, Herdman [19], observing that bacterial genome sizes clustered around multiples of 0.8Mb (i.e., 1.6Mb, 3.2Mb, etc.), suggested that the larger ones are the product of ancient duplications. The gene order of modern-day bacteria is not strong evidence for or against such duplication. There are often pairs of regions which are similar in gene content and order, but these are too rare and scattered to be convincing proof of a genome-wide duplication. If this event did occur, it has since been almost totally obscured by loss or divergence (in sequence and function) of one or both of the copies of most gene pairs, by lateral transfer of genes among related and unrelated organisms and by extensive rearrangement of the gene order. Nevertheless, prokaryotic genome duplication remains a possibility and often crops up in the literature, e.g., [23]. In contrast to plants, fungi, animals, and other eukaryotes which have a multiple-chromosome genome in their nucleii, prokaryotes tend to have a single, often circular, chromosome, so that translocation is not a possibility. They do not have meiosis, so genome duplication cannot arise as a result of a defect in this mechanism. It could, however, result from a fusion of two sister genomes. Reversal of long or short chromosomal segments is often cited as one of the predominant mechanisms for gene order rearrangement in unichromosomal genomes.

The prevalence and evolutionary importance of genome duplication, together with the fragmented nature of its present-day remnants, usually greatly obscured by subsequent developments at the sequence and chromosomal levels, lead to the question addressed in this paper: How can we reconstruct some or most of the original gene order at the time of genome duplication, based on traces conserved in the ordering

of those duplicate genes still identifiable? Solving this would allow us key insights into the mechanisms and consequences of this dramatic evolutionary event. A similar question can also be considered in the case of duplication of fragments of chromosomes [9].

Originally a duplicated genome contains two identical copies of each chromosome, but through intrachromosomal movements and reciprocal translocations, this simple doubled structure is disrupted. The problem considered here is therefore as follows: given a present-day genome modeled by a set of strings (chromosomes) of distinguished elements (genes), each gene appearing exactly twice in the genome, how to recover an ancestral duplicated genome by performing a minimal number of reversals and/or reciprocal translocations? We assume that a sign + or − is associated to each gene, representing its transcriptional orientation. Our method makes use of a formula of Hannenhalli and Pevzner (HP) for the classical problem of signed genome rearrangement.

In a series of papers published in 1995, HP solved the problems of calculating the minimum number of rearrangements necessary to transform one signed genome G into another signed genome H, with rearrangement models based on

- reversals only [17],
- translocations only [16], and
- both reversals and translocations [18].

Though the minimizing formulae and their derivations are different in each case, the frameworks for the three models are similar. They are based on a graph called the *breakpoint graph*, in which each vertex is incident to one black and one gray edge, black edges corresponding to genome G, and gray edges to genome H. This graph decomposes naturally into a set of color-alternating cycles. The number of cycles is the dominant term in the minimizing formulae. The other terms depend on overlap relationships among these cycles, and on their clustering into "good" and "bad" components.

The 1995 papers also presented exact polynomial algorithms for actually constructing a series of rearrangements satisfying the minimality criterion. Subsequently, many alternate versions have been proposed to make various parts of the algorithms more efficient [22, 6, 5, 37]. However, our results on duplicated genomes do not depend on these algorithms. After deriving an ancestral genome by our new methods, an efficient version of the HP algorithm can simply be applied to the present-day genome to convert it to the ancestral genome we obtain.

Our approach in this paper is, given a present-day genome G, to estimate its ancestral polyploid genome by one whose comparison with G minimizes the HP formulae. As the ancestral genome H is unknown, we can start only with the *partial graph* of black edges, and we must complete this graph with an optimal set of gray edges. Though the three evolutionary models described above have different aspects related to the particular kind of genome (multichromosomal or circular) and operation (translocations and/or reversals) considered, the key concepts are the same for the three models.

The first step of the general method is to complete the graph with "valid" gray edges, i.e., gray edges representing a duplicated genome, so as to maximize the number of cycles of the resulting graph. The key idea is to subdivide the graph into a set of disjoint subgraphs, called *natural and supernatural graphs*, that can be solved independently. This is detailed in section 5. These graphs first provide an upper bound for the number of cycles. This bound is presented in section 6. Section 7 then describes a linear algorithm, called *dedouble*, for constructing a completed graph, where

the number of cycles actually attains the upper bound. The main characteristic of *dedouble* is that any gray edge constructed links two vertices of the same supernatural graph. The second step of the general method consists in modifying *dedouble* in order to minimize the number of bad components. Though the concept of bad components is different for each of the three models, they are all related to the notion of *subpermutations* (SPs). Section 8 describes the general approach and the major modification to algorithm *dedouble*. Sections 9, 10, and 11 are then dedicated to the models with translocations only, translocations and reversals, and reversals only, respectively. Developments specific to each model are detailed in these sections. Finally, section 12 gives an application of our algorithm to the multichromosomal yeast genome, and section 13 gives another application to a circular mitochondrial genome.

We begin by formalizing the problem in the next section. We then introduce the HP graph and formulae in section 3 and introduce our notation and main definitions in section 4.

2. Formalizing the problem. We consider three models: translocations-only, both translocations and reversals, and reversals-only. The first two pertain to the multichromosomal context (eukaryotic nuclear genomes), while the third is relevant to single chromosome prokaryotic and organellar genomes.

A *string* is a sequence of signed (+ or −) terms (*genes*) from a set $\mathcal{B}$. A *multichromosomal genome* is a collection of at least two nonnull strings (*chromosomes*). For a string $X = x_1 x_2 \cdots x_r$, denote by $-X$ the *reverse* string $-x_r\ -x_{r-1} \cdots -\ x_1$.

In the models with translocations, a *rearranged duplicated genome* G is a multichromosomal genome containing an even number of chromosomes, such that each gene in $\mathcal{B}$ is present exactly twice, i.e., once in each of two different chromosomes, or twice in a single chromosome.

Example 1. Let $\mathcal{B} = \{a, b, c, d, e, f, g, h\}$ be a set of 8 genes, and let G be a genome consisting of four chromosomes:

$$1\text{: } +a\ +b\ -c\ +b\ -d; \quad 2\text{: } -c\ -a\ +f;$$
$$3\text{: } -e\ +g\ -f\ -d; \quad 4\text{: } +h\ +e\ -g\ +h.$$

G is a rearranged duplicated genome. Each gene appears exactly twice in the set of chromosomes; e.g., gene b appears twice in chromosome 1. Signs represent gene orientation.

A circular chromosome is a string $x_1 x_2 \cdots x_r$, where x_1 is considered to follow x_r. As most single chromosome genomes contain a circular chromosome, in this paper only these circular genomes are considered. However, the application of all the results to genomes with single noncircular chromosomes is straightforward.

In the reversals-only model, a rearranged duplicated genome consists of a single circular genome G containing each gene in $\mathcal{B}$ exactly twice.

Example 2. Let $G = +a\ +b\ -c\ +b\ -d\ -e\ +a\ +c\ -d\ -e$. G is a rearranged duplicated genome on the set of genes $\mathcal{B} = \{a, b, c, d, e\}$. That G is a circular genome means that vertex $+a$ is considered to follow vertex $-e$.

The problem is to calculate the minimum number of rearrangement operations required to transform a given rearranged duplicated genome G into some *perfect duplicated genome* H (or simply *duplicated genome*) to be found. We call this problem the *genome halving problem*. In the case of a multichromosomal genome, H consists of chromosomes $C_1, \ldots, C_{2N}$, where for each $i \in \{1, \ldots, 2N\}$, we have $C_i = C_j$ for exactly one $j \in \{1, \ldots, 2N\} \backslash \{i\}$. In the case of a circular genome, H is of the form $C\ C$ or $C\ -C$, where C is a string containing exactly one occurrence of each gene of $\mathcal{B}$.

Each of the three models permits a different combination of the rearrangement operations reversal and translocation. A *reversal* transforms some proper substring of a genome into its reverse. Let X_1, X_2, Y_1, and Y_2 be nonnull strings. A *reciprocal translocation* between two chromosomes $X = X_1X_2$ and $Y = Y_1Y_2$ is of the form $X_1X_2, Y_1Y_2 \longrightarrow X_1Y_2, Y_1X_2$ (prefix-prefix) or of the form $X_1X_2, Y_1Y_2 \longrightarrow X_1 - Y_1, -Y_2X_2$ (prefix-suffix) (see Figure 2.1).

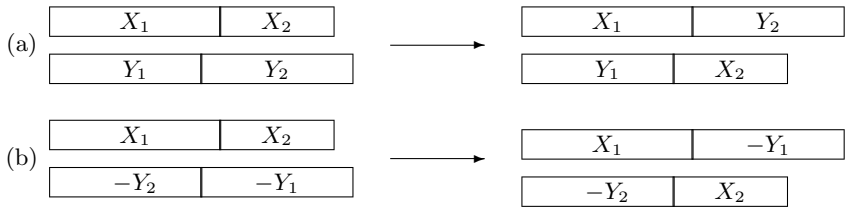


FIG. 2.1. *Reciprocal translocation between two chromosomes X_1X_2 and Y_1Y_2.* (a) *Prefix-prefix translocation.* (b) *Prefix-suffix translocation.*

3. The HP theory. Given two genomes H_1 and H_2 containing the same gene set $\mathcal{B}$, where each gene appears exactly once in each genome, the *genome rearrangement problem* is to find the minimum number of rearrangement operations necessary to transform H_1 into H_2 (or H_2 into H_1). HP designed polynomial algorithms for the reversals-only version of the problem (in the case of single chromosome genomes) [17], the translocations-only version [16], and the version with both reversals and translocations [18] (the latter two for multichromosomal genomes).

The algorithms all depend on a bicolored graph $\mathcal{G}_{12}$ constructed from H_1 and H_2. The details of this construction vary from model to model, due to the different ways chromosomal endpoints must be handled, but the general character of the graph is the same and may be summarized as follows.

Graph $\mathcal{G}_{12}$. If gene x of H_1 has positive sign, replace it by the pair x^tx^h, and if it is negative, replace it by x^hx^t. Then the vertices of $\mathcal{G}_{12}$ are just the x^t and the x^h for all x in $\mathcal{B}$. Any two vertices which are adjacent in some chromosome in H_1, other than x^t and x^h deriving from the same x, are connected by a black edge (thick lines in figures), and any two adjacent in H_2 are connected by a gray edge (thin lines). In the case of a single chromosome, the black edges may be displayed linearly according to the order of the genes in the chromosome (Figure 3.1). For a genome containing N chromosomes, N such linear orders are required (Figure 3.2), and the genes at either end of the chromosome must be treated somewhat differently.

Now, each vertex is incident to exactly one black and one gray edge so that there is a unique decomposition of $\mathcal{G}_{12}$ into c_{12} disjoint cycles of alternating edge colors. By the *size of a cycle* we mean the number of black edges it contains. Note that $c_{21} = c_{12} = c$ is maximized when $H_1 = H_2$, in which case each cycle has one black edge and one gray edge.

A rearrangement operation ρ, either a reversal or a translocation, is determined by the two points where it "cuts" the current genome which correspond to two black edges e and f. We say that ρ *is determined by the two black edges* e and f. Rearrangement operations may change the number of cycles of the graph so that minimizing the number of operations can be seen in terms of increasing the number of cycles as fast as possible. Let $\Delta(c)$ be the difference between the number of cycles before and after applying the rearrangement operation ρ. HP showed that $\Delta(c)$ may take on values

1, 0, or −1, in which cases they are called *ρ proper*, *improper*, or *bad*, respectively. Roughly speaking, an operation determined by two black edges in two different cycles will be bad, while one acting on two black edges within the same cycle may be proper or improper, depending on the type of cycle and the type of edges considered.

Key to the HP approach are the graph components. Two cycles, say, Cycles 1 and 2, all of whose black edges are related by the same linear order (i.e., are on the same line), and containing gray edges that "cross," e.g., gene i linked to gene j by a black edge (i.e., in H_1) in Cycle 1, gene k linked to gene t by a black edge in Cycle 2, but ordered i, k, j, t in H_2, are connected. A *component* of $\mathcal{G}_{12}$ is a maximal set of crossing cycles, excluding the case of a cycle of size 1 (see Figures 3.1 and 3.2). A component is termed *good* if it can be transformed to a set of cycles of size 1 by a series of proper operations, and *bad* otherwise. Bad components are called *subpermutations* in the translocations-only model, *hurdles* in the reversals-only model, and *knots* in the combined model. More details on bad components and how to solve them will be given in the sections dedicated to each of the three evolutionary models.

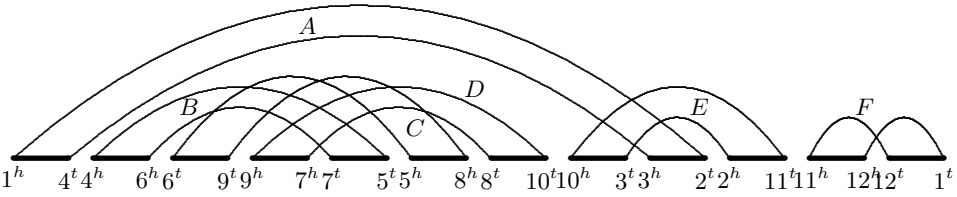


FIG. 3.1. *Graph $\mathcal{G}_{12}$ corresponding to circular genomes (i.e., the first gene is adjacent to the last gene) $H_1 = +1\,+4\,-6\,+9\,-7\,+5\,-8\,+10\,+3\,+2\,+11\,-12$ (black edges) and $H_2 = +1\,+2\,+3\cdots+12$ (gray edges). A, B, C, D, E, and F are the six cycles of $\mathcal{G}_{12}$. $\{A, E\}$, $\{B, C, D\}$, and $\{F\}$ are the three components of $\mathcal{G}_{12}$.*

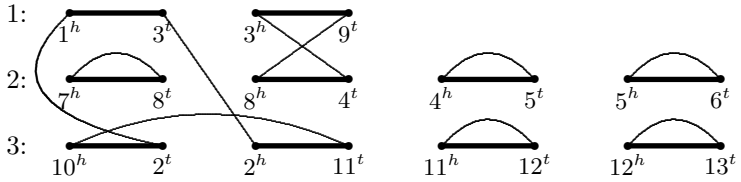


FIG. 3.2. *Graph $\mathcal{G}_{12}$ corresponding to genomes H_1, H_2, both with three chromosomes, where $H_1 = \{1 : 1\ 3\ 9\ ;\ 2 : 7\ 8\ 4\ 5\ 6\ ;\ 3 : 10\ 2\ 11\ 12\ 13\}$ and $H_2 = \{1 : 1\ 2\ 3\ 4\ 5\ 6\ ;\ 2 : 7\ 8\ 9\ ;\ 3 : 10\ 11\ 12\ 13\}$. All genes are signed "+." The edges, which are on the same horizontal row of the graph, correspond to a chromosome of H_1. There are seven cycles. As no cycle of size > 1 is contained on one row, $\mathcal{G}_{12}$ does not contain any component. Both genomes have the same set of endpoints, so we can omit the extremal vertices (x^t for initial genes and x^h for terminal genes) as discussed in section 4.*

The HP formulae for all three models may be summarized as follows:

$$\textbf{HP1}: \quad RO(H_1, H_2) = b(\mathcal{G}_{12}) - c(\mathcal{G}_{12}) + m(\mathcal{G}_{12}) + f(\mathcal{G}_{12}),$$

where $RO(G, H)$ is the minimum number of rearrangement operations (reversals and/or translocations), $b(\mathcal{G}_{12})$ is the number of black edges, $c(\mathcal{G}_{12})$ is the number of cycles, $m(\mathcal{G}_{12})$ is the number of bad components of $\mathcal{G}_{12}$, and $f(\mathcal{G}_{12})$ is a correction of size 0, 1, or 2 depending on the set of bad components.

Generally speaking, bad components are rare, so the number of cycles of $\mathcal{G}_{12}$ is the dominant parameter in the **HP1** formula if $b(\mathcal{G}_{12})$ is considered as a constant. In

other words, the more cycles there are, the fewer reversals we need to transform H_1 into H_2.

4. Preliminaries. To make use of the HP graph structure for our genome halving problem, we first introduce, arbitrarily, a distinction within each pair of identical genes in the rearranged duplicated genome G, labeling one occurrence x_1 and the other x_2 for each x in $\mathcal{B}$.

In the case of linear chromosomes (noncircular), the HP method requires that the two genomes being compared share the same set of chromosomal endpoints. To ensure this constraint for linear multichromosomal genomes, we add a new initial term O_{i1} and a new final term O_{i2} to each chromosome C_i. This also ensures that all translocations, including those which reduce (by *fusion*, e.g., null X_1Y_2, Figure 2.1) or augment (by *fission*, e.g., null X_1X_2, Figure 2.1) the number of chromosomes in the genome, can be treated as reciprocal translocations. This also allows us to consider genomes with an odd number $2N-1$ of chromosomes by adding a dummy chromosome consisting of just one initial and one final O, to obtain $2N$ chromosomes.

In each chromosome, each x_j (except the O_{ij}) is replaced by x_j^t and x_j^h as in the HP construction. Define

$$O = \{O_{i1}, O_{i2}\}_{i=1,\cdots,2N},\ V = \{x_j^s\}_{\substack{x\in\mathcal{B}\\ j=1,2}}^{s\in\{h,t\}},\ \mathbf{V} = O \cup V.$$

In the case of a circular genome, endpoints are irrelevant, and thus the set O is empty, and $\mathbf{V} = V$. We use the notation $\overline{1} = 2$, $\overline{2} = 1$, $\widetilde{t} = h$, $\widetilde{h} = t$. For $u = x_j^s \in V$, its *counterpart*, denoted $\overline{u}$, is $x_{\overline{j}}^s$ (the corresponding vertex in the paralogous gene), and its *obverse*, denoted $\widetilde{u}$, is $x_j^{\widetilde{s}}$ (the vertex corresponding to the other "end" of the gene). Note that $\overline{\overline{u}} = \widetilde{\widetilde{u}} = u$.

The *partial graph* $\mathcal{G}(\mathbf{V}, A)$ associated with G has the edge set A of black edges linking adjacent terms (other than the obverse) in G. The partial graph associated with the genome G of Example 1 is shown in Figure 4.1. To differentiate the two occurrences of each gene x, one is subscripted "1," and its counterpart is "2."

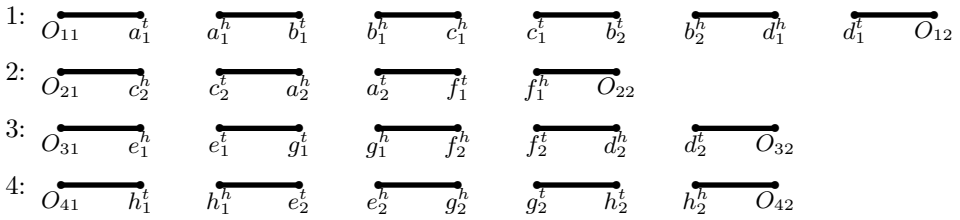


FIG. 4.1. *The partial graph* $\mathcal{G}(\mathbf{V}, A)$ *corresponding to Example* 1.

We are required to add to this partial graph a set Γ of gray edges so that every vertex in $\mathbf{V}$ is incident to exactly one black edge and one gray edge and so that the resulting genome is a perfectly duplicated one. A set Γ of gray edges giving rise to a duplicated genome is said to be *valid*. In the case of a multichromosomal genome, a chromosome of a perfectly duplicated genome should begin and end with two elements of O. The graph $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ obtained by adding a valid set Γ of gray edges is called a *completed graph* of $\mathcal{G}(\mathbf{V}, A)$. Lemmas 4.1 and 4.2 give the constraints that Γ should satisfy to be valid in the cases of multichromosomal and circular genomes, respectively.

LEMMA 4.1. *For multichromosomal genomes,* Γ *is valid if and only if the following conditions are satisfied:*

1. Γ *contains no edge of form* $(x, \overline{x})$ *for any* $x \in V$.
2. *Suppose* $(x, y) \in \Gamma$ *and* $y \in V$. *If* $x \in V$, *then* $(\overline{x}, \overline{y})$ *is also in* Γ. *Otherwise* $(x \in O)$, $\overline{y}$ *is also linked by a gray edge to an element of* O.
3. *The resulting genome does not contain any circular chromosome.*

Proof. Clearly, a duplicated genome must satisfy all three conditions. Suppose now that Γ is a set of gray edges so that every vertex of $\mathbf{V}$ is incident to exactly one gray edge, and Γ satisfies the three conditions. Then, from condition 3, as no circular fragment is present, and as the only "genes" with only one end are the elements of O, each chromosome of the resulting genome H has its two endpoints in O. From condition 1, the two copies of the same gene cannot be adjacent in H, and from condition 2, if two genes are adjacent in H, then their homologs are also adjacent in H in the same order. This ensures that each permutation (string) is present exactly twice in H. Therefore, H is a perfectly duplicated genome. ∎

LEMMA 4.2. *For circular genomes,* Γ *is valid if and only if the following conditions are satisfied:*

1. Γ *contains exactly zero or two edges of form* $(x, \overline{x})$.
2. *If* $(x, y) \in \Gamma$, *then* $(\overline{x}, \overline{y}) \in \Gamma$.
3. *The resulting genome consists of a single circular chromosome.*

Proof. The proof follows from the definition of a circular duplicated genome. ∎

To find a duplicated genome that gives rise to the minimal number of rearrangement operations, we have to construct a valid set of gray edges that minimizes the formula **HP1** (section 3). The key idea is to decompose the partial graph into a set of subgraphs that can be completed independently. We describe such a decomposition in the next section.

5. Decomposition into subgraphs. We define the set $\mathcal{NG}$ of *natural graphs* of $\mathcal{G}(\mathbf{V}, A)$ as follows.

DEFINITION 5.1. *Let* $e = (x, y) \in A$. *Define* A_e *recursively by* $(x, y) \in A_e$, *and if* $(x, y) \in A_e$, *then both the edge of* A *adjacent to* $\overline{x}$ *and the edge of* A *adjacent to* $\overline{y}$ *are also in* A_e.

Let $\mathbf{V}_e$ *be the subset of* $\mathbf{V}$ *made up of vertices incident to the edges in* A_e. *Then* $\mathcal{G}_e(\mathbf{V}_e, A_e)$ *is the* natural graph *(of* size $|A_e|$*) of* $\mathcal{G}(\mathbf{V}, A)$ *generated by* e. *Note that if* $f \in A_e$, *then* $A_f = A_e$.

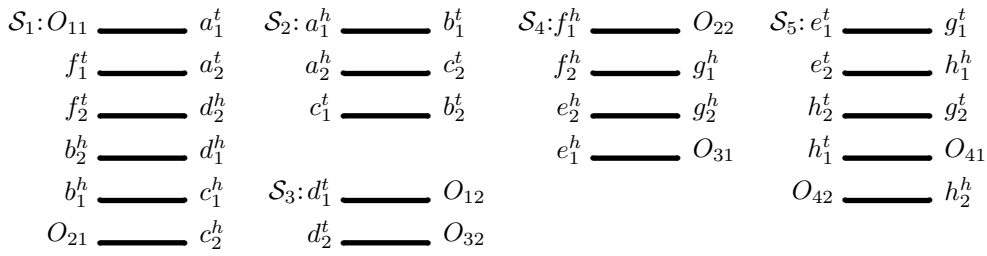


FIG. 5.1. *The natural graphs of the partial graph* $\mathcal{G}(\mathbf{V}, A)$ *of Figure* 4.1.

As an illustration, the decomposition of the partial graph of Figure 4.1 into natural graphs is given in Figure 5.1.

Let $\mathcal{G}_\alpha$ be a subgraph of $\mathcal{G}(\mathbf{V}, A)$. $\mathcal{G}_\alpha$ represents a set of fragments of the chromosomes of G. The subgraph $\mathcal{G}_\alpha$ is said to be *completable* if we can find a set of gray

edges linking the vertices of $\mathcal{G}_\alpha$ that gives rise to a set of fragments of a potential duplicated genome. Not every natural subgraph is completable. In the case of multichromosomal genomes, we proved in [10] that a natural graph is completable if and only if it is of even size or it contains vertices in O. Similarly, for circular genomes, all natural graphs of even size are completable. Moreover, as we can have at most two gray edges of form $(u, \overline{u})$, then at most two natural graphs of odd size are completable.

The underlying idea of the subdivision and amalgamating procedure is to form completable graphs. First, $\mathcal{NG}$ is subdivided into the following subsets:

- $\mathcal{NE}$ is the subset of $\mathcal{NG}$ containing the natural graphs of even size.
- $\mathcal{NO}$ is the subset of $\mathcal{NG}$ containing the natural graphs of odd size. We further subdivide $\mathcal{NO}$ into $\mathcal{NO}_+$ and $\mathcal{NO}_-$ according to whether the natural graphs include vertices in O or not. Note that $\mathcal{NO}_+$ may contain a natural graph formed by a single edge linking two vertices in O.

The set A contains $2(|\mathcal{B}| + N)$ edges in the case of multichromosomal genomes, and $2|\mathcal{B}|$ edges in the case of circular genomes. Moreover, the graphs of $\mathcal{NE}$ contain an even number of edges. Therefore, $\mathcal{NO}$ must also contain an even number of edges and thus an even number of graphs. We can then pair off all the graphs in $\mathcal{NO}$ as follows:

- Arbitrarily choose pairs of graphs in $\mathcal{NO}_+$ to amalgamate. The set of larger graphs thus formed is denoted $\mathcal{SO}_+$.
- Arbitrarily choose pairs of the remaining graphs in $\mathcal{NO}$ to amalgamate. This includes graphs in $\mathcal{NO}_-$ plus, if applicable, the remaining one in $\mathcal{NO}_+$. The set of graphs thus formed is denoted $\mathcal{SO}$.

We denote $\mathcal{SE} = \mathcal{NE} \cup \mathcal{SO}_+$, and we call the graphs of $\mathcal{SN} = \mathcal{SE} \cup \mathcal{SO}$ *supernatural*.

In the example of Figure 5.1, $\mathcal{NE} = \{\mathcal{S}_1,\ \mathcal{S}_3,\ \mathcal{S}_4\}$, $\mathcal{NO}_- = \{\mathcal{S}_2\}$, and $\mathcal{NO}_+ = \{\mathcal{S}_5\}$. Moreover, $\mathcal{SE} = \mathcal{NE}$, and if $\mathcal{S}_{25}$ is the supernatural graph obtained by amalgamating $\mathcal{S}_2$ and $\mathcal{S}_5$, then $\mathcal{SO} = \{\mathcal{S}_{25}\}$. The set $\{\mathcal{S}_1, \mathcal{S}_{25}, \mathcal{S}_3, \mathcal{S}_4\}$ is a decomposition of $\mathcal{G}(\mathbf{V}, A)$ into supernatural graphs.

Note that for circular genomes, $\mathcal{NO}_+$ is empty, and thus $\mathcal{NO}$ graphs are arbitrarily amalgamated. In that case, $\mathcal{SO}_+$ is empty and $\mathcal{SE} = \mathcal{NE}$.

Notation 1. In a supernatural graph $\mathcal{G}_\alpha(\mathbf{V}_\alpha, A_\alpha)$ of $\mathcal{NE} \cup \mathcal{SO}$, if a vertex $u \in \mathbf{V}_\alpha \cap O$ exists, then we denote by $\overline{u}$ the (only) other vertex in $\mathbf{V}_\alpha \cap O$. For example, in Figure 5.1, $\overline{0_{11}} = 0_{21}$, and $\overline{0_{41}} = 0_{42}$.

In a supernatural graph $\mathcal{G}_\alpha(\mathbf{V}_\alpha, A_\alpha)$ of $\mathcal{SO}_+$ made up of two natural graphs $\mathcal{G}_1(\mathbf{V}_1, A_1)$ and $\mathcal{G}_2(\mathbf{V}_2, A_2)$ of $\mathcal{NO}_+$, if $u \in \mathbf{V}_1 \cap O$, then we arbitrarily choose one of the two vertices of $\mathbf{V}_2 \cap O$ to be $\overline{u}$.

5.1. Ordering the edges of the natural subgraphs. To simplify the ensuing development, we use a particular representation of each supernatural graph $\mathcal{G}_\alpha(\mathbf{V}_\alpha, A_\alpha)$ of size $2n$, where $n > 1$. Relabeling the vertices in $\mathbf{V}_\alpha$ allows us to define a *suitable order* for the edges in A_α (cf. Figure 5.2).

1. If $\mathcal{G}_\alpha \in \mathcal{CE}$, $A_\alpha = \{e_1, e'_1, \ldots, e_n, e'_n\}$ such that the following hold:
   - $e_1 = (a_1, b_1)$; $e'_1 = (\overline{a_1}, b_2)$.
   - For each i, $1 < i < n$, $e_i = (a_i, \overline{b_{i-1}})$ and $e'_i = (\overline{a_i}, b_{i+1})$.
   - $e_n = (a_n, \overline{b_{n-1}})$; $e'_n = (\overline{a_n}, \overline{b_n})$.
2. If $\mathcal{G}_\alpha \in \mathcal{SO}$, let $\mathcal{G}_1(A_1)$ and $\mathcal{G}_2(A_2)$ be its two component natural subgraphs, where $A_\alpha = A_1 \cup A_2$. Then $A_1 = \{e_1, e'_1, \ldots, e_{n_1-1}, e'_{n_1-1}, e_{n_1}\}$, where e_i and e'_i are defined as above except that $e_{n_1} = (\overline{b_{n_1}}, \overline{b_{n_1-1}})$. Similarly, $A_2 = \{e_{n_1+1}, e'_{n_1+1}, \ldots, e_{n-1}, e'_{n-1}, e_n\}$ with $e_n = (\overline{b_n}, \overline{b_{n-1}})$.

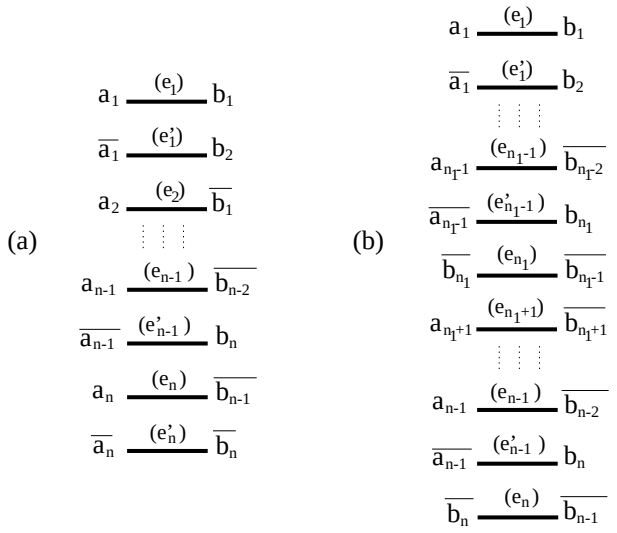


FIG. 5.2. *A suitable order for the edges of supernatural graphs.* (a) *A supernatural graph in* $\mathcal{SE}$. (b) *A supernatural graph in* $\mathcal{SO}$.

For an illustration, consider the supernatural graphs $\{\mathcal{S}_1, \mathcal{S}_{25}, \mathcal{S}_3, \mathcal{S}_4\}$ of our running example. By means of a relabeling of the vertices (a vertex x_1 could be relabeled as x_2, or vice-versa), one possible suitable order for the edges of the graphs is considered in Figure 7.5.

In the ensuing discussion, we start with any decomposition of $\mathcal{G}(\mathbf{V}, A)$ into a set $\mathcal{SN}$ of supernatural graphs in the suitable order.

As the dominant parameter in the **HP1** formula is the number of cycles, we begin by considering a set of valid gray edges maximizing the number of cycles of a completed graph. In the next section, we provide an upper bound on the number of cycles, and in section 7, we describe an algorithm for constructing a completed graph that allows us to reach this bound.

6. Upper bound on the number of cycles. We need a preliminary definition.

DEFINITION 6.1. *Let* $\mathcal{G}_\alpha(\mathbf{V}_\alpha, A_\alpha)$ *be a supernatural graph of size* $2n$. *Consider the ordering of* A_α *described in the last section. Then* $V_l = \bigcup_{1 \le i \le n} \{a_i, \overline{a_i}\}$ *is the set of* left vertices *of* $\mathbf{V}_\alpha$, *and* $V_r = \bigcup_{1 \le i \le n} \{b_i, \overline{b_i}\}$ *is the set of* right vertices *of* $\mathbf{V}_\alpha$.

Note that from the definition, a natural subgraph of $\mathcal{SO}$ has four more right vertices than left vertices.

The set $\mathbf{V}$ is partitioned into subsets of left and right vertices: x is a left vertex in $\mathbf{V}$ if it is a left vertex of a graph of $\mathcal{SN}$. Otherwise, it is a right vertex.

Let $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ be a completed graph of $\mathcal{G}(\mathbf{V}, A)$, and let C be a particular cycle of size r of the graph with vertex set $\mathbf{V}_C$ and black and gray edge sets A_C and Γ_C, respectively. We define the *signature* S_C of C to be the subset of $\mathbf{V}_C$ derived as follows: For every left vertex x in $\mathbf{V}_C$, if $\overline{x}$ is not already in S_C, then add x to S_C.

Let $\mathcal{S}$ be the set of signatures of all the cycles of $\mathcal{G}_\Gamma$. Define the *signature graph* with the set of nodes $\mathcal{S}$ and the set of edges E as follows: for all $S_1, S_2 \in \mathcal{S}$, S_1 and S_2 are linked by an edge in E if and only if there is a vertex x such that $x \in S_1$ and $\overline{x} \in S_2$.

In Figure 6.1, a completed graph is given on the left. It represents a completed

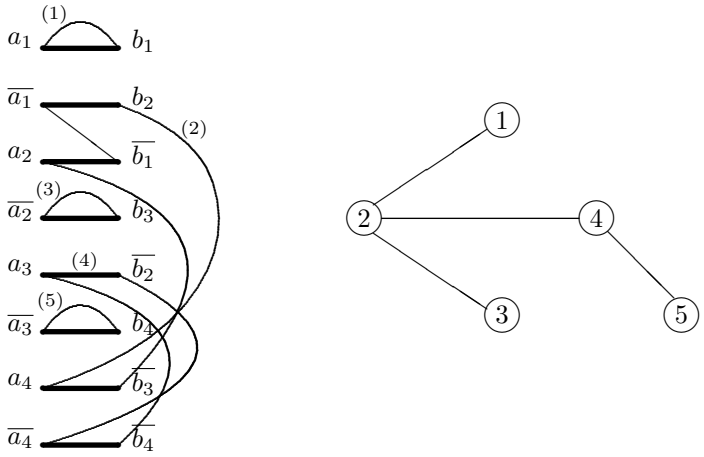


FIG. 6.1. *Example of a signature graph.*

supernatural graph of $\mathcal{SE}$. The completed graph is made up of five cycles, whose signatures are as follows:

$$1.\ \{a_1\};\quad 2.\ \{\overline{a_1}, a_2, a_4\};\quad 3.\ \{\overline{a_2}\};\quad 4.\ \{a_3, \overline{a_4}\};\quad 5.\ \{\overline{a_3}\}.$$

The graph on the right of Figure 6.1 is the signature graph derived from the graph on the left.

LEMMA 6.2. *In the case of multichromosomal genomes, the signature graph of any completed supernatural graph is connected.*

Proof. The proof is deduced from the fact that $\mathcal{SN}$ is a set of smallest completable graphs: for any supernatural graph $\mathcal{G}_\alpha$, there does not exist any subgraph of $\mathcal{G}_\alpha$ that is also completable. □

For a node S_C of $\mathcal{S}$, denote by $t(S_C)$ the number of vertices in S_C and by $\delta(S_C)$ the number of outgoing edges.

LEMMA 6.3. *For a multichromosomal genome, let $\mathcal{G}_e(\mathbf{V}_e, A_e)$ be a supernatural graph of size $2n$, where $n > 0$. Let $\mathcal{G}_e(\mathbf{V}_e, A_e, \Gamma_e)$ be a completed graph, and let c_e be its number of cycles. If $\mathcal{G}_e \in \mathcal{SE}$, then $c_e \leq n+1$. Otherwise ($\mathcal{G}_e \in \mathcal{SO}$), $c_e \leq n$.*

Proof. Let $\mathcal{S}$ be the set of vertices and E the set of edges of the signature graph of $\mathcal{G}_e(\mathbf{V}_e, A_e, \Gamma_e)$. Then $c_e = |\mathcal{S}|$.

For every $S_C \in \mathcal{S}$, $\delta(S_C) \leq t(S_C)$. Now $\sum_{S_C \in \mathcal{S}} t(S_C) \leq 2n$ so that

$$|E| = \frac{1}{2} \sum_{S_C \in \mathcal{S}} \delta(S_C) \leq \frac{1}{2} \sum_{S_C \in \mathcal{S}} t(S_C) \leq n.$$

From Lemma 6.2, a signature graph is connected so that $|\mathcal{S}| \leq |E| + 1 \leq n + 1$.

For the case $\mathcal{G}_e(\mathbf{V}_e, A_e) \in \mathcal{SO}$, $\sum_{S_c \in \mathcal{S}} t(S_C) \leq 2n - 2$. By the same argument as above,

$$|\mathcal{S}| \leq |E| + 1 = \frac{1}{2} \sum_{S_C \in \mathcal{S}} \delta(S_C) + 1 \leq \frac{1}{2} \sum_{S_C \in \mathcal{S}} t(S_C) + 1 \leq n. \qquad \square$$

Results are slightly different for circular genomes.

LEMMA 6.4. *In the case of circular genomes, the signature graph of any completed supernatural graph of $\mathcal{SE}$ is connected. On the other hand, at most one completed*

supernatural graph of $\mathcal{SO}$ has a signature graph with two connected components. The signature graph corresponding to any other graph of $\mathcal{SO}$ is connected.

Proof. In the case of circular genomes, at most two natural graphs among all natural graphs of odd size are completable. This follows from the fact that a circular genome contains at most two adjacencies of form $(x, \overline{x})$. Therefore, as a supernatural graph of $\mathcal{SO}$ is obtained by concatenating two natural graphs of odd size, at most one completable supernatural graph of $\mathcal{SO}$ has a signature graph with two connected components. Any other graph of $\mathcal{SN}$ cannot be subdivided into smallest completable graphs and thus have signature graphs reduced to one connected component. ∎

LEMMA 6.5. *For a circular genome, let $\mathcal{G}_e(\mathbf{V}_e, A_e, \Gamma_e)$ be a completed supernatural graph of size $2n$, and let c_e be its number of cycles. If $\mathcal{G}_e \in \mathcal{SE}$, then $c_e \leq n+1$. Moreover, there is at most one supernatural graph $\mathcal{G}_e$ of $\mathcal{SO}$ such that $c_e = n+1$. For all the other supernatural graphs of $\mathcal{SO}$, $c_e \leq n$.*

Proof. The proof is similar to that of Lemma 6.3 but uses the result of Lemma 6.4. ∎

Notation 2. We denote by $\boldsymbol{\gamma(G)}$ the number of "good" supernatural graphs:

- In the case of a multichromosomal genome G, $\gamma(G) = |\mathcal{SE}|$.
- In the case of a circular genome G, if $\mathcal{SO}$ is empty, then $\gamma(G) = |\mathcal{SE}|$; otherwise, $\gamma(G) = |\mathcal{SE}| + 1$.

THEOREM 6.6. *Let $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ be a completed graph of $\mathcal{G}(\mathbf{V}, A)$, and let $c(\mathcal{G}_\Gamma)$ be its number of cycles. Then*

$$c(\mathcal{G}_\Gamma) \leq \frac{|A|}{2} + \gamma(G).$$

Proof. If any cycle C of $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ is "good," i.e., such that all black edges of C belong to the same supernatural graph of $\mathcal{SG}$, then, according to Lemmas 6.3 and 6.5, $c(\mathcal{G}_\Gamma) \leq \frac{|A|}{2} + \gamma(G)$.

Suppose now that there exist "bad cycles" in $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$, i.e., cycles containing black edges of different supernatural graphs. Let c_b be the number of bad cycles, and let c_g be the number of good cycles of $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$. Then $c(\mathcal{G}_\Gamma) = c_b + c_g$.

Let $\mathcal{G}_p(\mathbf{V}_p, A_p)$ be a supernatural graph, and let $\mathcal{C}_p$ be the set of cycles of $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ containing at least one edge in A_p. Let c_{g_p} be the number of good cycles and c_{b_p} the number of bad cycles of $\mathcal{C}_p$. Denote by $\{x_i, 1 \leq i \leq |\mathbf{V}_p|\}$ the set of vertices of $\mathbf{V}_p$.

Suppose that C is a bad cycle of $\mathcal{C}_p$ of size > 1. Denote $C = x_1x_2 - -x_3x_4 - - x_5x_6, \ldots$, where x_i's are the vertices in $\mathbf{V}_p$ and "$--$" denote paths in the cycle that do not contain any vertex in $\mathbf{V}_p$. Some of these paths can be empty.

We modify the bad cycles of $\mathcal{C}_p$ by the following procedure:

1. For any bad cycle $C = x_1x_2 - -x_3x_4 - -x_5x_6 \cdots$ and any x_i with an even i, do
2. If $x_{i+1} \neq \overline{x_i}$, do
3. Remove the gray edges adjacent to x_i, x_{i+1}, $\overline{x_i}$, $\overline{x_{i+1}}$;
4. Construct the gray edges (x_i, x_{i+1}) and $(\overline{x_i}, \overline{x_{i+1}})$;
5. Else, there is another path of form $x_j - -\overline{x_j}$ (i.e., $x_{j+1} = \overline{x_j}$) either in C, or in another cycle of $\mathcal{C}_p$;
6. Choose such a path, if possible in C, otherwise in another bad cycle, else, in a good cycle;
7. Remove the gray edges adjacent to x_i, x_{i+1}, x_j, and x_{j+1};
8. Construct the gray edges (x_i, x_j), (x_{i+1}, y_{j+1}), $(\overline{x_i}, \overline{x_j})$, and $(\overline{x_{i+1}}, \overline{x_{j+1}})$;
9. End of If
10. End of For

The procedure constructs a completed graph $\mathcal{G}_p(\mathbf{V}_p, A_p, \Gamma_p)$. Let c_p be the number of cycles of $\mathcal{G}_p(\mathbf{V}_p, A_p, \Gamma_p)$. As the only way to decrease the number of cycles is to amalgamate pairs of bad cycles or to amalgamate at most once a bad cycle with a good one (lines 5 to 8 of the procedure), we have $c_p \geq c_{g_p} + c_{b_p} - \lceil \frac{c_{b_p}}{2} \rceil \geq c_{g_p} + \lfloor \frac{c_{b_p}}{2} \rfloor$. Let $c_{max,p}$ be the maximal number of cycles of a completed graph of $\mathcal{G}_p(\mathbf{V}_p, A_p)$. Then $c_{g_p} + \lfloor \frac{c_{b_p}}{2} \rfloor \leq c_{max,p}$.

Let now c_{max} be the maximal number of cycles of a completed graph of $\mathcal{G}(\mathbf{V}, A)$, and let c_d be the total number of bad cycles of $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$. Then (1) $c_g + c_d \leq c_{max} + \lfloor \frac{c_d}{2} \rfloor \Longrightarrow c_g + \lceil \frac{c_d}{2} \rceil \leq c_{max}$.

On the other hand, as any bad cycle of a supernatural graph (a cycle containing at least one edge in the supernatural graph) corresponds to a bad cycle of another supernatural graph, the total number of cycles of $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ is (2) $c(\mathcal{G}_\Gamma) = c_g + c_b \leq c_g + \lceil \frac{c_d}{2} \rceil$. We deduce from inequalities (1) and (2) that $c(\mathcal{G}_\Gamma) \leq c_{max} \leq \frac{|A|}{2} + \gamma(G)$. □

7. Maximizing the number of cycles. Based on the decomposition of $\mathcal{G}(\mathbf{V}, A)$ into supernatural graphs, can we construct a completed graph $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ having $c(\mathcal{G}_\Gamma) = \gamma(G) + \frac{|A|}{2}$ cycles? By Theorem 6.6, this would necessarily be a *maximal completed graph*, that is, a completed graph with a maximal number of cycles. In this section, we focus on multichromosomal genomes. Modifications that have to be introduced in the case of circular genomes are presented in section 11.

We will use the following notation: for any set U of natural graphs, we denote by $\mathbf{V}_U$ the set of vertices of all natural graphs of U and by A_U the set of all black edges of U. For example, $\mathbf{V}_{\mathcal{SE}}$ will be the set of vertices of $\mathcal{SE}$.

We require a preliminary definition. A *fragment* of a genome is just a linear substring of G. For example, $F_1 = +g_1\ -f_2\ -d_2$ and $F_2 = 0_{11}\ +a_1\ +b_1$ are two fragments of the genome represented by the partial graph of Figure 4.1. A fragment has two *endpoints*, unless it is restricted to one element of O. In the example given here, the two endpoints of F_1 are g_1^t and d_2^t, and the two endpoints of F_2 are 0_{11} and b_1^h. We call a fragment that has its two endpoints in V a $\mathcal{B}$-fragment.

Suppose that we have reached a certain step s in the construction, that Γ_s is the set of gray edges already constructed, and that $\mathcal{G}(\Gamma_s)$ is the "partially completed" graph obtained at this step. Suppose also that the natural graph being considered at this step is $\mathcal{G}_\alpha$, that the set of gray edges linking vertices of $\mathcal{G}_\alpha$ already constructed is $\Gamma_{s,\alpha}$, and that $\mathcal{G}_\alpha(\Gamma_{s,\alpha})$ is the obtained "partially completed" natural graph. A vertex of V is said to be *unlinked* if it is not yet linked by a gray edge at the current step of the algorithm.

We denote by $\mathcal{F}$ the fragments set resulting from Γ_s. At the outset, $\mathcal{F}$ is made up of the *unitary fragments*, which include not only $x^t x^h$ for all $x \in \mathcal{B}$ (the *$\mathcal{B}$-unitary fragments*) but also the $2N$ elements of O (the *O-unitary fragments*). As the construction proceeds, whenever a gray edge (x, y) is created, the fragment containing x and the one containing y are joined together.

DEFINITION 7.1. *Let $\mathcal{V}_s$ be a subset of the set of unlinked vertices at step s of the algorithm. The* border *of $\mathcal{V}_s$ is the set of all vertices x of $\mathcal{V}_s$ such that $x \in O$, or x is an endpoint of a $\mathcal{B}$-fragment $F \in \mathcal{F}$, and the second endpoint of F is not in $\mathcal{V}_s$.*

The graph $\mathcal{G}(\Gamma_s)$ is *bad* if there exists a subset U of $\mathcal{SN}$ such that the border of $\mathbf{V}_{s,U}$ is empty, where $\mathbf{V}_{s,U}$ is the set of unlinked vertices of $\mathbf{V}_U$ at step s. Otherwise, $\mathcal{G}(\Gamma_s)$ is a *good graph*. For an example, see Figure 7.1.

LEMMA 7.2. *Any set of gray edges linking the remaining unlinked vertices of a bad graph creates at least one circular fragment.*

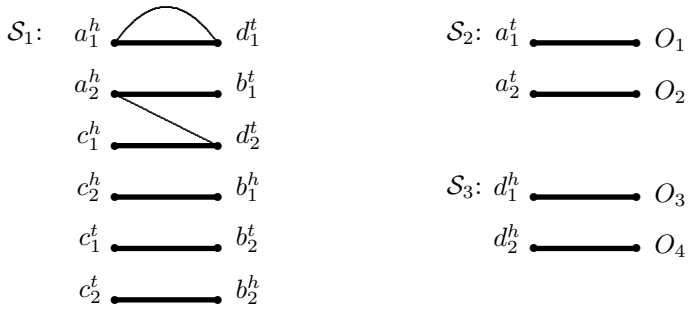


FIG. 7.1. *The partial graph corresponding to the genome with the two chromosomes:* $O_1 + a_1 + d_1\ O_3$; $O_2 + a_2 + b_1 - c_2 - b_2 + c_1 + d_2\ O_4$. *If we construct the two gray edges* $(a_1^h, d_1^t), (a_2^h, d_2^t)$, *the graph becomes bad, as the border of the supernatural graph* $\mathcal{S}_1$ *becomes empty.*

Proof. Suppose that U is a subset of $\mathcal{SN}$ such that the border of $\mathbf{V}_{s,U}$ is empty. Then there is a set $\mathcal{F}_d$ of fragments such that the set of endpoints of $\mathcal{F}_d$ is exactly $\mathbf{V}_{s,U}$. Then, by linking the vertices of $V_{s,U}$ by gray edges, all we can do is close all the fragments of $\mathcal{F}_d$, that is, create at least one circular fragment. □

The above lemma implies that we have to be careful during the execution of the algorithm so as not to end up with a bad graph. Now suppose that $\mathcal{G}(\Gamma_s)$ is a good graph. Let $x, y, \overline{x}, \overline{y}$ be four unlinked vertices of $\mathcal{G}_\alpha(\Gamma_{s,\alpha})$. The pair of "potential" gray edges $\{(x, y), (\overline{x}, \overline{y})\}$ will be termed *impossible* if, when constructed, it creates either a circular fragment or a bad graph and *possible* otherwise. It is easy to see that a pair of edges $\{(x, y), (\overline{x}, \overline{y})\}$ creates a circular fragment if and only if one of the following properties is satisfied (see Figure 7.2).

Property I. The vertices $\{x, y\}$ are the endpoints of a $\mathcal{B}$-fragment of $\mathcal{F}$.

Property II. The pairs of vertices $\{x, \overline{y}\}, \{\overline{x}, y\}$ are the endpoints of two $\mathcal{B}$-fragments of $\mathcal{F}$.

FIG. 7.2. *The left (resp., right) figure represents Property* I *(resp., Property* II*). Bold lines represent fragments, and thin lines represent the "potential" gray edges* $(x, y), (\overline{x}, \overline{y})$. *In any of these cases, the resulting fragment is circular.*

Now, let us consider a third property of a pair $\{(x, y), (\overline{x}, \overline{y})\}$ of potential gray edges.

Property III. x, y are two endpoints of two different fragments F_1, F_2 of $\mathcal{F}$, and neither one of the two other endpoints of F_1, F_2, if any, is in $\mathcal{G}_\alpha$.

LEMMA 7.3. *Suppose that* $\mathcal{G}(\Gamma_s)$ *is good. Suppose that, at step* $s+1$, *we construct the two gray edges* $(x, y), (\overline{x}, \overline{y})$. *If these gray edges do not satisfy Property* III, *then* $\mathcal{G}(\Gamma_{s+1})$ *is good.*

Proof. Let F_1, F_2 be the two fragments such that x is an endpoint of F_1 and y is an endpoint of F_2. Suppose that x, y do not satisfy Property III. Let U be any subset of $\mathcal{SN}$.

- Suppose F_1, F_2 have four endpoints, and all of these endpoints are in $\mathcal{G}_\alpha$. Then it is easy to see that linking F_1 to F_2 does not modify either the border

corresponding to $\mathcal{G}_\alpha$ or that corresponding to U. Thus the state of $\mathcal{G}_\alpha$ (good or bad) could not have changed between steps s and $s+1$.

- Suppose that F_1, F_2 have at least three endpoints, and three such endpoints are in $\mathcal{G}_\alpha$. The subgraph U is bad if and only if $\mathbf{V}_{U,s+1}$ contains the fourth endpoint of F_1, F_2 not in $\mathcal{G}_\alpha$ and the border of $\mathbf{V}_{U,s+1}$ is empty. However, in that case, U would also have been bad at step s, which is a contradiction. ∎

For example, in Figure 7.1, the two gray edges $(a_2^h, b_1^t), (a_1^h, b_2^t)$ do not satisfy Property III, as the second endpoint of the fragment containing b_1^t is b_1^h, and b_1^h is in $\mathcal{S}_1$. Therefore, constructing these gray edges does not create a bad graph.

COROLLARY 7.4. *If a pair of potential gray edges* $\{(x,y),(\overline{x},\overline{y})\}$ *of a good graph does not satisfy any of the Properties* I, II, *and* III, *then it is a possible pair of gray edges.*

Let x be an unlinked vertex of $\mathcal{G}_\alpha$. Then x is one of the two endpoints of a path (made up of a succession of black and gray edges) completely contained in $\mathcal{G}_\alpha$. We denote by x^c the second endpoint of this path. We say that a gray edge *closes the path* if and only if it links x to x^c.

Algorithm *dedouble* described in Figure 7.3 completes each supernatural graph of $\mathcal{SN}$, one after the other, in a specific order. The notation and edge order are those described in section 5.1.

LEMMA 7.5. *At each step, algorithm* dedouble *constructs possible pairs of gray edges.*

Proof. Supernatural graphs of $\mathcal{SE}$, *with* $n = 1$. At the beginning of the algorithm, the gray edges $(a_1, b_1), (\overline{a_1}, \overline{b_1})$ of $\mathcal{SE}$ are clearly possible, as they form fragments of the original genome G (Figure 7.4.(a)).

Suppose that we have reached a certain step in the construction and that the current supernatural graph of $\mathcal{SO}$ has the four vertices $b_1, b_2, \overline{b_1}, \overline{b_2}$ (Figure 7.4.(b)).

Suppose b_1, b_2 do not satisfy Property I, that is, they are the two endpoints of a fragment $F = b_1 \cdots b_2$. F cannot be a fragment of G, as in that case G would contain a circular fragment. Thus F should contain an adjacency (b_i, b_j) constructed from a supernatural subgraph of $\mathcal{SO}$, which means that $(a_i, \overline{a_i})$ and $(a_j, \overline{a_j})$ are two adjacencies in G. Then, if $F = b_1 \cdots a_i a_j \cdots b_2$, it is easy to see that G should contain two fragments of form $b_1 \cdots b_i \overline{b_i} \cdots \overline{b_1}$ and $b_2 \cdots b_j \overline{b_j} \cdots \overline{b_2}$. But since $(b_1, \overline{b_1}), (b_2, \overline{b_2})$ are two adjacencies in G (from the fact that the four vertices belong to a graph in $\mathcal{SO}$), this implies that G contains two circular fragments, which is impossible. Therefore, (b_1, b_2) (or, similarly, $(\overline{b_1}, \overline{b_2})$) does not create a circular fragment. We can prove in a similar way that $(b_1, b_2), (\overline{b_1}, \overline{b_2})$ do not satisfy Property II.

Suppose now that (b_1, b_2) creates a bad graph. Then there exists a subset U of $\mathcal{SN}$ such that the border of $\mathbf{V}_{U,s}$ is $B(U,s) = \{b_1, b_2, \overline{b_1}, \overline{b_2}\}$. This implies that the vertices of $\mathbf{V}_{U,s}$ belong to two fragments of G with the four endpoints $\{b_1, b_2, \overline{b_1}, \overline{b_2}\}$. This is also impossible as the two edges $(b_1, b_2), (\overline{b_1}, \overline{b_2})$ would give rise to a circular fragment in G.

Therefore, at the end of step 1 of the algorithm, the partial graph obtained is a good graph.

Supernatural graphs of $\mathcal{SE}$, *with* $n > 1$. Suppose that we have reached a good graph $\mathcal{G}(\Gamma_s)$ with a certain number of completed supernatural graphs. Suppose that $\mathcal{G}_\alpha$ is the supernatural graph currently being completed and that the current vertices to be considered are $a_i, \overline{a_i}$. Suppose first that $i \leq n-2$. It is easy to see, from the construction, that $a_i^c \neq \overline{a_i}^c$ and thus the two pairs of gray edges $p_{i,1}$ and $p_{i,2}$ are different.

Algorithm dedouble:

Subgraphs in $\mathcal{SE}$, $n = 1$

1. Construct the gray edges $\{(a_1, b_1), (\overline{a_1}, \overline{b_1})\}$ (cf. Figure 7.4.(a));

Subgraphs in $\mathcal{SO}$, $n = 1$

2. Construct the gray edges $\{(b_1, b_2), (\overline{b_1}, \overline{b_2})\}$ (cf. Figure 7.4.(b));

Subgraphs in $\mathcal{SE}$, $n > 1$

3. **For** $i = 1$ to $n - 2$ **Do**
4. Set $c = a_i^c$ and $d = \overline{a_i}^c$;
5. **If** $p_{i,1} = \{(a_i, c), (\overline{a_i}, \overline{c})\}$ does not satisfy Properties I, II, and, III, **Then**
6. Construct the gray edges of $p_{i,1}$;
7. **Else**
8. Construct the gray edges of $p_{i,2} = \{(\overline{a_i}, d), (a_i, \overline{d})\}$;
9. **End of if**
10. **End for**
11. Set $c = a_{n-1}^c$ and $d = \overline{a_{n-1}}^c$ (cf. Figure 7.4.(c));
12. **If** $p_{n-1,1} = \{(a_{n-1}, c), (\overline{a_{n-1}}, \overline{c})\}$ and $p_{n,1} = \{(a_n, d), (\overline{a_n}, \overline{d})\}$ do not
13. satisfy any of the Properties I, II, and III, **Then**
14. Construct the gray edges of $p_{n-1,1}, p_{n,1}$;
15. **Else**
16. Construct the gray edges of $p_{n-1,2} = \{(\overline{a_{n-1}}, d), (a_{n-1}, \overline{d})\}$ and
17. $p_{n,2} = \{(a_n, \overline{c}), (\overline{a_n}, c)\}$;
18. **End of if**

Subgraph $\mathcal{G}_\alpha$ in $\mathcal{SO}$, $n > 1$

Let $\mathcal{G}_1, \mathcal{G}_2$ be the two natural graphs amalgamated to form $\mathcal{G}_\alpha$, and $n_1 > 1$;

19. **For** $i = 1$ to $n_1 - 2$ **Do**
20. Construct gray edges as in the previous case;
21. **End of for**
22. **For** $i = n_1 + 1$ to $n - 1$ **Do**
23. Construct gray edges as in the previous case;
24. **End of for**
25. Set $c = a_{n_1-1}^c, d = \overline{a_{n_1-1}}^c$ (cf. Figure 7.4.(d));
26. Let $e, \overline{e}$ be the only unlinked vertices in $\mathcal{G}_2$;
27. **If** $p_{n-1,1} = \{(a_{n_1-1}, c), (\overline{a_{n-1}}, \overline{c})\}$ and $p_{n,1} = \{(e, d), (\overline{e}, \overline{d})\}$ do not
28. satisfy any of the Properties I, II, and III, **Then**
29. Construct the gray edges of $p_{n-1,1}, p_{n,1}$;
30. **Else**
31. Construct the gray edges of $p_{n-1,2} = \{(\overline{a_{n_1-1}}, d), (a_{n_1-1}, \overline{d})\}$ and
32. $p_{n,2} = \{(c, e), (\overline{c}, \overline{e})\}$;
33. **End of if**

FIG. 7.3. *Algorithm for constructing a maximal completed graph.*

Suppose now that $p_{i,1}$ is impossible. We want to prove that $p_{i,2}$ is possible. Suppose $p_{i,1}$ satisfies Property I. That means that a_i and c are the endpoints of the same fragment F. Therefore, a_i and $\overline{d}$ cannot be the endpoints of the same fragment, which means that $p_{i,1}$ does not satisfy Property I. The vertices a_i and d are not the endpoints of the same fragment either, which means that $p_{i,2}$ does not satisfy

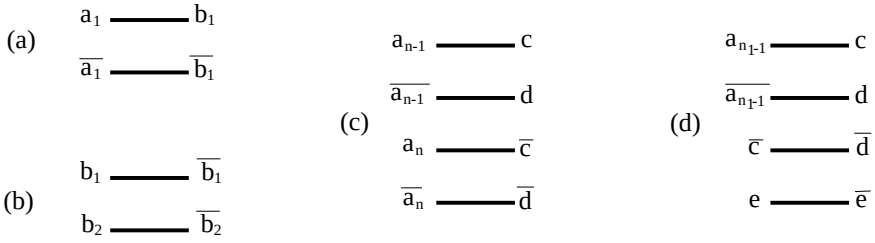


FIG. 7.4. *Different situations considered by* dedouble. (a) *A supernatural graph of* $\mathcal{SE}$*, with* $n = 1$. (b) *A supernatural graph of* $\mathcal{SO}$*, with* $n = 1$. (c) *The last step for a supernatural graph of* $\mathcal{SE}$*; corresponds to lines* 10 *to* 15 *of the algorithm.* (d) *The last step for a supernatural graph of* $\mathcal{SO}$*; corresponds to lines* 23 *to* 29 *of the algorithm. In* (c) *and* (d)*, edges represent paths that can contain more than one edge.*

Property II. Now, since a_i and $\overline{d}$ are two endpoints of two fragments, one of them, which is F having both endpoints in $\mathcal{G}_\alpha$, $p_{i,2}$, does not satisfy Property III either.

We prove similarly that, if $p_{i,1}$ satisfies Property II, then $p_{i,1}$ cannot satisfy any of the three properties.

Suppose now that $p_{i,1}$ creates a bad graph. That means that there exists a subset U of $\mathcal{E}$ such that the border of $V_{U,s}$ is $B(U,s) = \{a_i, \overline{a_i}, c, \overline{c}\}$; then a_i and c should belong to two different fragments with the two other endpoints not in $\mathcal{G}_\alpha$. Then clearly $p_{i,2}$ cannot satisfy Property I or Property II. Suppose that it satisfies Property III. That means that there exists a subset U' of $\mathcal{E}$ such that the border of $V_{U',s}$ is $B(U',s) = \{a_i, \overline{a_i}, d, \overline{d}\}$. Therefore, the border of $U \cup U'$ is restricted to $\{a_i, \overline{a_i}\}$ and is of size 2. The other vertices of $U \cup U'$ cannot be in O (as, otherwise, these vertices would have been part of the border), and if u is in $U \cup U'$, then $\overline{u}, u^s, \overline{u}^s$ are also in $U \cup U'$. Therefore, the number of vertices of $U \cup U'$ is $4m + 2$ for some m. However, this is impossible as the number of vertices of $U \cup U'$ remaining unlinked should be divisible by 4.

To finish the proof, we have to show that, if $p_{n-1,1}$ and $p_{n,1}$ are impossible, then $p_{n-1,2}$ and $p_{n,2}$ are possible (Figure 7.4(c)).

Suppose (a_{n-1}, c) (and $(\overline{a_{n-1}}, \overline{c})$) satisfies Property I, that is, a_{n-1} and c are the endpoints of a fragment F. Then clearly neither $(\overline{a_{n-1}}, d)$ nor $(\overline{a_n}, c)$ satisfies Property I or Property II. Suppose $(\overline{a_{n-1}}, d), (\overline{a_n}, c)$ give rise to one circular fragment. This is possible if a_{n-1}, c and $\overline{a_n}, d$ are the endpoints of two fragments. However, in that case, the supernatural graph $\mathcal{G}_\alpha$ would have had an empty border just before the current step, and the graph would have been a bad graph. However, this contradicts the recurrence hypothesis. Suppose finally that $p_{n-1,2}$ and $p_{n,2}$ create a bad graph. Then there exists a subset U of $\mathcal{SN}$ such that the border of U is in $\{a_{n-1}, \overline{a_{n-1}}, a_n, \overline{a_n}, c, \overline{c}, d, \overline{d}\}$. However, just before this step, $U \cup \{\mathcal{G}_\alpha\}$ would have been a bad graph, which contradicts the recurrence hypothesis.

The remaining cases are treated in a similar way, and we prove with similar arguments that, in any of these cases, $p_{n-1,2}, p_{n,2}$ are possible.

Supernatural graphs of $\mathcal{SO}$ *with* $n > 1$. The construction method for $1 \leq i \leq n_1 - 2$ and $n_1 + 1 \leq i \leq n - 1$ is identical to that in a supernatural subgraph of $\mathcal{SE}$ for $1 \leq i \leq n - 2$. Therefore, the same proof as before holds in that case. Finally, we should prove that, if $p_{n-1,1}$ and $p_{n,1}$ are impossible, then $p_{n-1,2}$ and $p_{n,2}$ are possible. To do so, arguments similar to those for a supernatural subgraph of $\mathcal{SE}$ are used to treat each case. □

Example 3. Consider the genome G of Example 1 and the decomposition of its

partial graph into the supernatural graphs $\{\mathcal{S}_1, \mathcal{S}_{25}, \mathcal{S}_3, \mathcal{S}_4\}$. Figure 7.5 depicts the completed graph produced by *dedouble*.

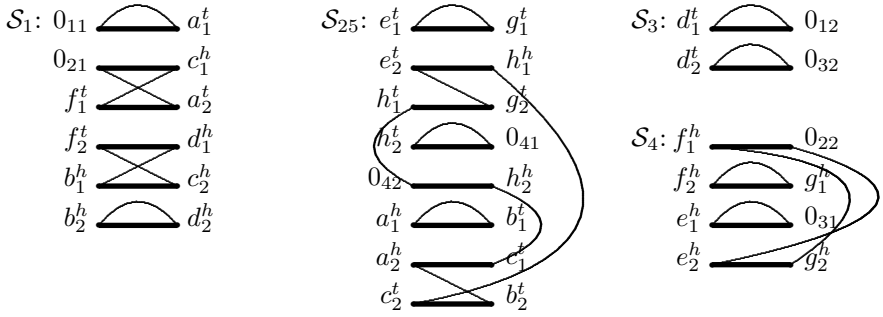


FIG. 7.5. *The completed graph* $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ *constructed by* dedouble.

The number of cycles in the completed graph is $c(\mathcal{G}) = 13$. As $\gamma(G) = 3$ and $|A| = 20$, according to Theorem 6.6, it is a maximal completed graph.

The corresponding duplicated genome H is made up of the four chromosomes

1. $O_{11} + a_1 + b_1 - d_1\ O_{12}$;
2. $O_{21} + a_2 + b_2 - d_2\ O_{32}$;
3. $O_{42} + h_1 + c_2 + f_2 - g_1 + e_1\ O_{31}$;
4. $O_{41} + h_2 + c_1 + f_1 - g_2 + e_2\ O_{22}$.

THEOREM 7.6. *Algorithm* dedouble *constructs a maximal completed graph* $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$*, containing* $c(G) = \frac{|A|}{2} + \gamma(G)$ *cycles.*

Proof. To prove this result, it is sufficient to prove that, for every supernatural graph $\mathcal{G}_\alpha$ with $2n$ black edges, if $\mathcal{G}_\alpha \in \mathcal{SE}$, then the number of cycles in the completed graph obtained by *dedouble* is $c(\mathcal{G}_\alpha) = n+1$, and if $\mathcal{G}_\alpha \in \mathcal{SO}$, this number is $c(\mathcal{G}_\alpha) = n$.

Let $\mathcal{G}_\alpha$ be a supernatural graph of $\mathcal{SE}$. For each i, $1 \leq i \leq n-2$, the algorithm constructs either (a_i, a_i^c) or $(\overline{a_i}, \overline{a_i}^c)$. Thus, at each step of the construction, at least one path is closed to form a cycle. Finally, it is easy to see, from Figure 7.4(c), that instructions 11–16 close three more cycles. Therefore, in total, at least $n+1$ cycles are formed in $\mathcal{G}_\alpha$. According to Lemma 6.3, the maximal number of cycles of a completed graph of $\mathcal{SE}$ is $n+1$. Therefore, $c(\mathcal{G}_\alpha) = n+1$.

Similarly, for a supernatural graph $\mathcal{G}_\alpha$ of $\mathcal{SO}$ with $2n$ black edges, steps 17–22 of the algorithm close at least $n-2$ cycles. Then it is easy to see, from Figure 7.4(d), that instructions 24–29 close two more cycles. Therefore, as n is the maximal number of cycles of a completed graph of $\mathcal{SO}$ (Lemma 6.3), $c(\mathcal{G}_\alpha) = n$. ∎

The following theorem is a direct consequence of Theorems 6.6 and 7.6.

THEOREM 7.7. *The number of cycles of a maximal completed graph of* $\mathcal{G}(\mathbf{V}, A)$ *is*

$$c(G) = \frac{|A|}{2} + \gamma(G).$$

Complexity. At each step, algorithm *dedouble* considers at most four black edges of the graph and constructs two gray edges with four vertices of the considered black edges. Choosing the right vertices to connect requires checking Properties I, II, and III for at most two pairs of gray edges. This is clearly done in constant time. Thus each step of the algorithm takes constant time. As each step constructs two gray edges, the graph is completed in $\frac{|A|}{2}$ steps. Therefore, the time complexity of algorithm *dedouble* is $O(|A|)$.

8. Bad components. We turn now our attention to minimizing the number of bad components of a completed graph. Even if the concept of bad components is different for each of the three models considered in this paper (translocations-only, reversals-only, or both reversals and translocations), it is always related to the notion of "subpermutation" introduced by Hannenhalli [16] and summarized below.

Given two genomes H_1 and H_2 containing the same gene set, where each gene appears exactly once in each genome, a subpermutation of H_1 (or, similarly, of the breakpoint graph $\mathcal{G}_{12}$ associated with H_1 and H_2) is a subsequence $S = u_1u_2, \cdots u_{p-1}u_p$ of a chromosome X of H_1 such that there is a permutation P and a subsequence $T = P(S) = u_1v_2 \cdots v_{p-1}u_p$ of a chromosome Y of H_2, with $v_2 \neq u_2$ and $v_{p-1} \neq u_{p-1}$. A *minimal SP* (minSP) is an SP not containing any other SP, and a *maximal SP* (maxSP) is an SP not included in any other SP.

We call *the interval of a component* C the interval $I = [u_l, u_r]$, where u_l and u_r are the endpoints of C. The interval I is such that no gray edge links a vertex of I to a vertex outside of I, and at least one cycle of I is of size greater than 1. A *minimal component* is a component whose interval contains no other component. There is a bijection between the SPs of $\mathcal{G}_{12}$ and the components of $\mathcal{G}_{12}$. More precisely, let S be an SP, let $\Pi = \{\pi_1, \ldots, \pi_p\}$ be the set of components containing the vertices of S, and for any i, let $\mathbf{V_i}$ be the set of vertices of π_i. Then the following hold:

- S_i is an SP contained in S (inner SP of S, possibly S itself) if and only if S_i corresponds to an interval of a component π_i of Π. We call this component the *component of the SP* S_i.
- S_i is a minimal inner SP of S if and only if S_i corresponds to an interval of a minimal component of Π.

Example 4. Consider the following two circular genomes and the corresponding breakpoint graph (Figure 8.1):

$$G = +a_1\ +b_1\ +c_1\ +d_1\ +e_1\ +d_2\ -f_1\ -e_2\ -f_2\ +a_2\ -b_2\ +c_2,$$
$$H = +a_1\ +b_1\ +c_1\ +d_1\ +e_1\ +f_2\ -f_1\ -e_2\ -d_2\ -c_2\ -b_2\ -a_2.$$

Each of the three components of this graph is made up of a single cycle.

$\mathcal{C}_1$ is the component of the SP $S_1 = +e_1\ +d_2\ -f_1\ -e_2\ -f_2\ +a_2\ -b_2\ +c_2\ +a_1$.

$\mathcal{C}_2$ is the component of the SP $S_2 = +d_2\ -f_1\ -e_2\ -f_2$.

$\mathcal{C}_3$ is the component of the SP $S_3 = +a_2\ -b_2\ +c_2$.

The only two minSPs are S_2 and S_3.

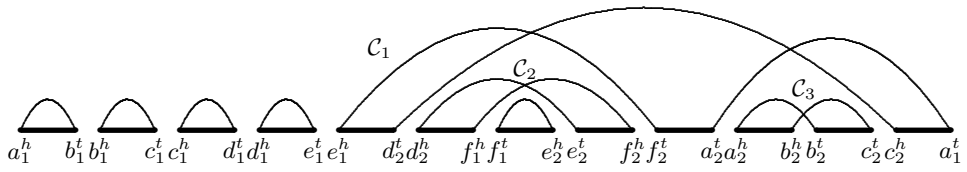


FIG. 8.1. *Breakpoint graph corresponding to genomes G and H.*

For the problem of rearrangement by translocations [16], all minSPs are bad components of an HP graph. More precisely, if $s(\mathcal{G}_{12})$ is the number of minSPs of $\mathcal{G}_{12}$, then, in formulae **HP1** (section 3), $m(\mathcal{G}_{12}) = s(\mathcal{G}_{12})$. For the problem of rearrangement by reversals, or by reversals and translocations, certain SPs can still be solved by proper operations, while others, the "bad components," require bad operations to be solved. The *hurdles* in the case of reversals [17] and the *knots* in the

case of reversals and translocations [18] are the bad (intrachromosomal) minSPs and maxSPs.

Returning to our genome halving problem, we want to determine the minimal number of such (bad) SPs in a completed graph of $\mathcal{G}(\mathbf{V}, A)$. In the case of circular genomes, we need to distinguish between SPs that do not contain both x and $\overline{x}$ for the same vertex x, which we call *normal*, and those that do, the *special* ones. As duplicated multichromosomal genomes cannot have both x and $\overline{x}$ on one chromosome, all SPs are normal for multichromosomal genomes. In the rest of the paper, if not specified, an SP will designate a normal one.

DEFINITION 8.1. *Let $S = x_1 x_2 \cdots x_{n-1} x_n$ be a subsequence of a chromosome of G. S is a* local SP *of G if S is a* real local SP *or a* potential local SP*, namely:*

- *S is a real local SP of G if $\{x_1, \ldots, x_n\} \cap O = \emptyset$ and there exists another subsequence of a chromosome of G of form $\overline{S} = \overline{x_1} P(\overline{x_2}, \ldots, \overline{x_{n-1}})\overline{x_n}$, where P is a permutation other than the identity.*
- *S is a potential local SP if either* i. *$\{x_1, x_n\} \subset O$, and there exists a chromosome of G containing a subsequence $\overline{S} = O_1 P(\overline{x_2}, \ldots, \overline{x_{n-1}})O_2$, where P is a permutation other than the identity and $\{O_1, O_2\} \in O$, or* ii. *$x_1 \in O$, and there exists a chromosome containing a subsequence $\overline{S} = O_1 P(\overline{x_2}, \ldots, \overline{x_{n-1}})x_n$, where P is a permutation other than the identity and $O_1 \in O$. An analogous condition holds for x_n.*

We call $\overline{S}$ the complementary sequence *of S. We say that a local SP (real or potential) S is* minimal *if it does not contain any subsequence corresponding to another local SP.*

For circular genomes, as the notion of endpoints is irrelevant, potential SPs do not exist and all local SPs are real ones.

Example 5. Let $G = +a_1\ +b_1\ +c_1\ +d_1\ +e_1\ -d_2\ +b_1\ +c_1\ -a_2\ +e_2$. The subsequence $S = +a_1 +b_1 +c_1 +d_1$ is a local SP of G. In the genome G of Example 4, the subsequence $+a_1 +b_1 +c_1$ is a local SP of G.

8.1. Correcting the completed graph obtained by algorithm *dedouble*. In this section, we describe a modification of algorithm *dedouble* that will be used to produce an optimal completed graph (i.e., a completed graph giving rise to a duplicated genome minimizing the rearrangement distance to G).

Let $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ be a maximal completed graph produced by *dedouble*, and let $S = x_1 \cdots x_n$ be an SP of $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$. The following procedure applies to the SP S.

Procedure spoil-SP(S):

Remove all the edges of Γ adjacent to the vertices of $\{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$;
Construct the edges (x_k, x_{k+1}) and $(\overline{x_k}, \overline{x_{k+1}})$ for all k, $1 \leq k < n$.

Consider the maximal completed graph $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ produced by *dedouble*. Let $\mathcal{S}$ be the set of SPs in $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ that do not correspond to local SPs of G. Correcting $\mathcal{G}_\Gamma$ consists in applying *spoil-SP* to each $S \in \mathcal{S}$.

Complexity. To correct the completed graph $\mathcal{G}$ produced by *dedouble*, we have to consider all the SPs of $\mathcal{G}$. This problem is equivalent to that of decomposing a breakpoint graph into its components. As shown in [22], this can be done in time $O(|A|)$. Then, verifying if an SP is a local SP of G and applying procedure *spoil-SP* takes time linear in the number of edges of the considered SP. Therefore, the total time needed to correct the graph is linear in the number of black edges $|A|$ of the graph. As algorithm *dedouble* has also been shown to be linear in $|A|$, the complexity of the whole algorithm (*dedouble* and graph correction) is $O(|A|)$.

8.2. Genome with no local SP. In this section, we show that for a genome with no local SP, the corrected graph is optimal.

LEMMA 8.2. *Suppose that the completed graph* $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ *produced by* dedouble *contains an SP* S*. If* S *is not a local SP of* G*, then* spoil-SP*(*S*)* *gives rise to a completed graph* $\mathcal{G}_{\Gamma'}(\mathbf{V}, A, \Gamma')$ *containing at least the same number of cycles as* $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ *and one less SP.*

Proof. Let $\mathcal{C}$ be the set of cycles of $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ containing at least one vertex in $V(x, \overline{x}) = \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$, $\mathcal{C}(x)$ the subset of $\mathcal{C}$ containing cycles with at least one vertex in $V(x) = \{x_1, \ldots, x_n\}$, and $\mathcal{C}(\overline{x})$ the subset of $\mathcal{C}$ containing cycles with at least one vertex in $V(\overline{x}) = \{\overline{x_1}, \ldots, \overline{x_n}\}$. Let $c = |\mathcal{C}|$, $c(x) = |\mathcal{C}(x)|$, and $c(\overline{x}) = |\mathcal{C}(\overline{x})|$. As S is an SP, $\mathcal{C}(x) \cap \mathcal{C}(\overline{x}) = \emptyset$.

Let $\mathcal{CP}$ be the set of cycles pairs (C_k, C_l) of $\mathcal{C}(x)$ such that $C_k \neq C_l$, and there is a vertex x_k in C_k and a vertex x_l in C_l such that $\overline{x_k}$ and $\overline{x_l}$ belong to the same cycle in $\mathcal{C}(\overline{x})$. Let $\pi = |\mathcal{CP}|$.

There are at most $\frac{n}{2} - \pi$ cycles in $\mathcal{C}(\overline{x})$, so $c \leq c(x) + \frac{n}{2} - \pi$.

Let $\mathcal{G}_{\Gamma'}(\mathbf{V}, A, \Gamma')$ be the graph obtained after applying procedure *spoil-SP*$(x_1, \ldots, x_n)$, and let $\mathcal{C}'$, $\mathcal{C}'(x)$ and $\mathcal{C}'(\overline{x})$ be the sets defined respectively as $\mathcal{C}$, $\mathcal{C}(x)$ and $\mathcal{C}(\overline{x})$ but for $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma')$. Let $c' = |\mathcal{C}'|$, $c'(x) = |\mathcal{C}'(x)|$, and $c'(\overline{x}) = |\mathcal{C}'(\overline{x})|$. As only size 1 cycles are formed with $V(x)$ vertices, $c'(x) = \frac{n}{2}$.

Let $C_1, \ldots, C_m$ be the cycles of size > 1 of $\mathcal{C}(x)$ for all r, $1 \leq r \leq m$, $|C_r| = p_r$, and let $\{x_{r_1}, \ldots, x_{r_{q_r}}\}$ be the vertices of $V(x)$ contained in C_r. Suppose we transform C_1 into $\frac{p_1}{2}$ size 1 cycles. The vertices $\{\overline{x_{1_1}}, \ldots, \overline{x_{1_{q_1}}}\}$ belong to at least one cycle. Then transform C_2 into $\frac{n}{2}$ size 1 cycles. If $(C_1, C_2) \in \mathcal{CP}$, then the vertices $\{\overline{x_{1_1}}, \ldots, \overline{x_{1_{q_1}}}, \overline{x_{2_1}}, \ldots, \overline{x_{2_{q_2}}}\}$ belong to at least 2 cycles; otherwise, they belong to at least 1 cycle.

By continuing this reasoning until C_m, we show that $c'(\overline{x}) \geq c(x) - \pi$. Thus $c' \geq \frac{n}{2} + c(x) - \pi$, and so $c \leq c'$. The completed graph $\mathcal{G}_{\Gamma'}(\mathbf{V}, A, \Gamma')$ is then also maximal but no longer contains the SP $x_1 \cdots x_n$.

Suppose that the procedure *spoil-SP*$(x_1 \cdots x_n)$ creates an SP that was not in $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$. Since the only modified edges are those linking vertices of $V(x, \overline{x})$, this new SP has to be formed by vertices in $V(\overline{x})$. Thus $x_1 \cdots x_n$ is necessarily a local SP of G. □

As a corollary to Lemma 8.2 we have the following.

COROLLARY 8.3. *For a genome* G *with no local SP, the corrected graph produced by* dedouble *is maximal and contains no SP.*

Then, from the formula **HP1** (section 3) and from the fact that, for all three rearrangement models considered, a bad component is attached to an SP of the graph (section 8), if G is a genome with no local SP, then the corrected graph produced by *dedouble* is optimal (i.e., gives rise to a duplicated genome minimizing the rearrangement distance to G).

In the remainder of this paper, it will be implicit that the correction of the graph, as described in the previous section, is incorporated at the end of algorithm *dedouble*. We turn next to the case in which G contains local SPs.

8.3. General formula for the rearrangement distance. The next lemma shows that any maximal completed graph should contain at least as many SPs as the number of real local SPs of G.

LEMMA 8.4. *Suppose that* G *contains a real local SP* $S = x_1 \cdots x_n$*. Suppose that the completed graph* $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ *contains no SP made up of the vertices* $\{x_1, \ldots, x_n\}$.

If c_{max} is the maximal number of cycles of a completed graph of $\mathcal{G}(\mathbf{V}, A)$ and $c(\mathcal{G}_\Gamma)$ is the number of cycles of $\mathcal{G}_\Gamma$, then $c(\mathcal{G}_\Gamma) \leq c_{max} - 2$.

Proof. Let $X_S = \{x_1, \ldots, x_n\}$ be the vertices of a subsequence $S = x_1 \cdots x_n$ of a certain chromosome of G, and let $\overline{X_S} = \{\overline{x_1}, \ldots, \overline{x_n}\}$ be the vertices of the complementary sequence $\overline{S}$ contained in a chromosome of G (another one or the same).

Let $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ be a maximal completed graph. Suppose that some vertices in X_S are linked by gray edges to vertices outside X_S. Let X be this set of vertices. Vertices in X are of two types: those linked to vertices in $\overline{X}$ and those linked to vertices outside $X \cup \overline{X}$. Denote $X_1 = \{x_{k_1}, \ldots, x_{k_l}\}$ as the set of l vertices of the first type, $X_2 = \{x_{p_1}, \ldots, x_{p_m}\}$ as the set of m vertices of the second type, and $Y = \{y_1, \ldots, y_m\} \subset \mathbf{V} \setminus X \cup \overline{X}$ as the vertices adjacent to them.

As all X vertices are adjacent to each other by black edges, a cycle containing a vertex in $X \cup Y$ contains at least two vertices of this set. Thus at most $\frac{l+m}{2}$ cycles contain a vertex in $X \cup Y$. Similarly, at most $\frac{m}{2}$ cycles contain a vertex in $\overline{X_2} \cup \overline{Y}$. Moreover, a cycle containing a vertex in $\overline{X_1}$ should contain a vertex in X_1. Therefore, the number of cycles containing a vertex in $X \cup Y \cup \overline{XY}$ is at most $\frac{l+m}{2} + \frac{m}{2} = m + \frac{l}{2}$.

Now, let $\mathcal{G}_{\Gamma'}(\mathbf{V}, A, \Gamma')$ be the completed graph obtained from $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ by the following procedure:

For all $x \in X$ do
- Remove gray edges adjacent to x and $\overline{x}$;
- Construct the gray edge (x, x'), where x' is the vertex in X linked to x by a black edge;
- Construct the gray edge $(\overline{x}, \overline{x'})$;

For all $y \in Y$ do
- Construct the gray edge (y, y'), where y' is the vertex in $\mathbf{V} \setminus X$ linked to y by a black edge;
- Construct the gray edge $(\overline{y}, \overline{y'})$.

Then exactly $\frac{l+m}{2}$ cycles have vertices in X and they are all of size 1, and exactly $\frac{m}{2}$ cycles have vertices in Y and they are also of size 1. Moreover, there is no cycle containing at the same time a vertex in $\overline{X}$ and another one in $\overline{Y}$, and there are at least two cycles with a vertex in $\overline{X}$ or a vertex in $\overline{Y}$. Therefore, the number of cycles containing vertices in $X \cup Y \cup \overline{XY}$ is at least $\frac{l+m}{2} + \frac{m}{2} + 2 = m + \frac{l}{2} + 2$. As the above procedure does not modify the other cycles, $\mathcal{G}_{\Gamma'}(\mathbf{V}, A, \Gamma')$ has at least two more cycles than $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$, which is a contradiction with the fact that $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ is a maximal completed graph. □

Remark 1. Let S be a local SP of G and $\overline{S}$ the complementary sequence of S. We can suppose, without loss of generality, that *dedouble* constructs only cycles of size 1 with vertices of $\overline{S}$ and that the SPs of the final completed graph are formed by the vertices of S. (We can always modify the resulting completed graph so that it satisfies these properties.)

For multichromosomal genomes, potential SPs give rise to additional problems. The goal is to minimize the number of such potential SPs that become SPs of the final completed graph. For circular genomes, all local SPs are real ones. However, in that case, one additional problem is due to special SPs.

Let $RO(G)$ be the minimal number of rearrangement operations required to transform G into a duplicated genome. Though $RO(G)$ is different depending on the model

considered (reversals, translocations, reversals and translocations), we will prove in the coming sections that all results can be summarized by the following formula:

$$RO(G) = \frac{|A|}{2} - \gamma(G) + m(G) + \phi(G),$$

where $m(G)$ is the number of bad real local SPs of G and $\phi(G)$ is a correction depending on bad potential local SPs (for multichromosomal genomes) and special SPs (for circular genomes).

Moreover, we will show that, with an appropriate construction of natural graphs, and with other minor corrections in the case of sorting by translocations and reversals, the completed graph produced by algorithm *dedouble* gives rise to the rearrangement distance.

9. Genome halving with translocations only. For two multichromosomal genomes H_1 and H_2, if $\mathcal{G}_{12}$ is the breakpoint graph associated to H_1 and H_2, the minimal number $T(H_1, H_2)$ of translocations required to transform H_1 to H_2 is given by the formulae proved in [16]:

$$\textbf{HP2}: \quad T(H_1, H_2) = b(\mathcal{G}_{12}) - c(\mathcal{G}_{12}) + s(\mathcal{G}_{12}) + f(\mathcal{G}_{12}),$$

where $s(\mathcal{G}_{12})$ is the number of minSPs of H_1. In other words, in the formula **HP1**, we have $m(\mathcal{G}_{12}) = s(\mathcal{G}_{12})$.

The value of the parameter $f(\mathcal{G}_{12})$ depends on a characteristic of the breakpoint graph, defined in [16]. The graph $\mathcal{G}_{12}$ has an *even-isolation* if the next three conditions are satisfied:

1. All minSPs of $\mathcal{G}_{12}$ are on a single chromosome of H_1.
2. $s(\mathcal{G}_{12})$ is even.
3. All minSPs are contained within a single SP.

If $\mathcal{G}_{12}$ has an even-isolation, then $f(\mathcal{G}_{12}) = 2$; if $\mathcal{G}_{12}$ has an odd number of minSPs, then $f(\mathcal{G}_{12}) = 1$; otherwise, $f(\mathcal{G}_{12}) = 0$ [16].

Returning to our problem of genome halving, denote by $\mathbf{T}(\mathbf{G})$ the minimal number of translocations required to transform G into a perfectly duplicated genome. In section 7, we described an algorithm for constructing a maximal completed graph in the case of multichromosomal genomes. We also proved, in section 8, that the minimal number of SPs of a completed graph can be deduced from the local (real or potential) SPs of G. The following corollary is a direct consequence of these results (Theorem 7.7, Corollary 8.3, and formula **HP2**).

COROLLARY 9.1. *If G does not contain any local SP, then* $T(G) = |A| - c(G) = \frac{|A|}{2} - \gamma(G)$.

Moreover, in section 8, we treated the case of real SPs. It remains now to consider potential local SPs.

Let S be a potential SP with two ends O_1, O_1' in O, and let O_2, O_2' be the two ends of $\overline{S}$. S becomes a real SP if and only if $\overline{O_2} = O_1$ and $\overline{O_2'} = O_1'$. Similarly, let S be a potential SP with only one end O_1 in O, and let O_2 be the vertex of $\overline{S}$ in O. S becomes a real SP if and only if $\overline{O_2} = O_1$. The problem is to avoid such situations.

According to formula **HP2**, we need only to minimize the number of minSPs of a completed graph (instead of SPs). Therefore, we consider only potential local minSPs. In the ensuing discussion, we just call them potential SPs.

Remark 2. Let S be a potential SP and $V(S, \overline{S})$ the set of vertices of S and $\overline{S}$ excluding those in O. The number of natural graphs containing vertices both in

$V(S, \overline{S})$ and O is exactly two if S has both its ends in O, and one if S has only one end in O. We call these graphs the *graphs associated to* S.

We distinguish between two kinds of potential SPs.

DEFINITION 9.2. *A potential SP is* even *(PES) if its associated graphs (one or two) are in* $\mathcal{NE}$*, i.e., are of even size. Otherwise, the potential SP is* odd *(POS). A POS necessarily has both its ends in* O *and thus two associated graphs in* $\mathcal{NO}_+$.

Notation 3. We denote $\mathcal{PES}$ the set of all PES, and $e = |\mathcal{PES}|$. For i, $1 \leq i \leq e$, P_i is the set of (one or two) graphs associated to the ith PES for an arbitrary ordering of the PESs.

We denote $\mathcal{POS}$ the set of all POS, and $o = |\mathcal{POS}|$. For every i, $1 \leq i \leq o$, denote by $Q_i = (A_i, A'_i)$ the pair of graphs associated to the ith POS for an arbitrary ordering of the POSs.

In section 5, we arbitrarily amalgamated pairs of natural graphs of odd size to form supernatural graphs. To avoid transforming a POS into an SP, we introduce a more deterministic way to amalgamate graphs of $\mathcal{POS}$. If $|\mathcal{POS}| > 1$, we proceed as follows:

Procedure amalgamating POS. For every i, $1 \leq i \leq o$, amalgamate each graph of the pair Q_i with a graph of a pair Q_j, where $j \neq i$.

Similar constraints are required in amalgamating PESs to avoid transforming them into SPs. If $|\mathcal{PES}| > 1$, we proceed as follows:

Procedure amalgamating PES. For every i, $1 \leq i \leq e$, amalgamate each graph of P_i with a graph in P_j, where $j \neq i$. Moreover, if $\mathcal{PES}$ has at least one P_i with two graphs and if a last nonamalgamated graph G_P remains, then G_P should belong to a P_i of size 2. Suppose G_1 and G_2 are amalgamated, O_1 and O'_1 are the two vertices of $G_1 \cap O$, and O_2 and O'_2 are the two vertices of $G_2 \cap O$. Then set $\overline{O_1} = O_2$ and $\overline{O'_1} = O'_2$.

Note that, in the case of the PESs, we amalgamate even size (completable) natural graphs. The consequence is that *dedouble*, applied to such supernatural graphs, generates a completed graph that is no longer maximal. This gives rise to additional difficulties.

After amalgamating the graphs of $\mathcal{PES} \cup \mathcal{POS}$ by the procedures described above, there remain some nonamalgamated graphs. This gives rise to eight possible configurations. For some of them, additional graphs are amalgamated.

C1. There remain no nonamalgamated graphs.

C2. There remains one Q_i in $\mathcal{POS}$. This happens when $\mathcal{POS}$ contains a single POS.

C3. There remains one P_i of two graphs in $\mathcal{PES}$. This happens when $\mathcal{PES}$ contains a single PES.

C4. There remains one graph in $\mathcal{PES}$, and it belongs to a P_i of size 2.

C5. There remains one graph in $\mathcal{PES}$, and it belongs to a P_i of size 1. This happens when all P_is are of size 1 and e is even.

C6. There remains one $Q_i = (G_1, G_2)$ in $\mathcal{POS}$ and one G_3 in $\mathcal{PES}$ belonging to a P_i of size 1. Then we amalgamate the three graphs G_1, G_2, and G_3 if that does not create an even-isolation. If O_1 and O'_1 are the vertices of $G_1 \cap O$, O_2 and O'_2 are the vertices of $G_2 \cap O$, and O_3 and O'_3 are the vertices of $G_3 \cap O$, then we set $\overline{O_1} = O_3$, $\overline{O'_1} = O'_2$, and $\overline{O_2} = O'_3$.

C6' will denote the configuration that would give rise to an even-isolation. In this case, the graphs are not amalgamated.

C7. There remains one Q_i in $\mathcal{POS}$ and one graph in $\mathcal{PES}$ belonging to a P_i of size 2.

C8. There remains one $Q_i = (G_1, G_2)$ in $\mathcal{POS}$ and one $P_i = (G_3, G_4)$ in $\mathcal{PES}$. Then we amalgamate G_1 and G_2 and one of the two graphs of P_i if that does not create an even-isolation. Counterpart elements are set similarly to C6. C8' will denote the configuration that would give rise to an even-isolation. In that case, the graphs are not amalgamated.

A local SP that is either real or potential, but not solved by the amalgamating procedure described above, is called a *final SP*.

In the remainder of this section, $\mathcal{SG}$ will designate the set of completable graphs obtained by the procedure described above for the graphs in $\mathcal{POS} \cup \mathcal{PES}$, and by the usual way (section 5) for the other natural graphs.

Notation 4. Consider the following parameters:

- $\mathbf{s}(\mathbf{G})$ is the number of real minSPs of G;
- $\mathbf{sp}(\mathbf{G})$ is the number of graphs obtained by amalgamating $\mathcal{PES}$ graphs;
- $\psi(\mathbf{G}) = 1$ if configuration C6 or C8 is encountered, and $\psi(\mathbf{G}) = 0$ otherwise;
- $\mathbf{sr}(\mathbf{G}) = 0$ if one of the configurations C1, C4, C6, or C8 is encountered; $sr(G) = 1$ for C2, C3, C5, or C7; $sr(G) = 2$ for C6' or C8'. $sr(G)$ is the number of potential SPs that become final SPs.
- $\mathbf{f}(\mathbf{G}) = 2$ if the set of final SPs represents an even-isolation; $f(G) = 1$ if the number of final SPs is odd; $f(G) = 0$ otherwise.

Recall that $c(G)$ is the number of cycles of a maximal completed graph of $\mathcal{G}(\mathbf{V}, A)$ (Theorem 7.6).

THEOREM 9.3. *Let $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ be the completed graph produced by* dedouble. *Let H be the resulting duplicated genome. Then $c(\mathcal{G}_\Gamma) = c(G) - sp(G) - \psi(G)$, $s(\mathcal{G}_\Gamma) = s(G) + sr(G)$, and*

$$T(G, H) = |A| - c(G) + sp(G) + \psi(G) + s(G) + sr(G) + f(G).$$

The minimal number of translocations required to transform G into a duplicated genome is $T(G) = T(G, H)$.

Proof. According to Corollary 8.3 and Lemma 8.4, if G does not contain any PESs, then *dedouble* produces a maximal completed graph $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ with $c(G)$ cycles.

Suppose now that G contains local SPs. Let $\mathcal{G}_3$ be a graph of $\mathcal{SG}$ obtained by amalgamating two natural graphs of $\mathcal{PES}$: $\mathcal{G}_1$ of size n_1 and $\mathcal{G}_2$ of size n_2. Given that this graph has as many left edges as right edges, a proof similar to that of Lemma 6.3 shows that the maximal number of cycles of a completed graph of $\mathcal{G}_3$ is $\frac{n_1+n_2}{2} + 1$, and *dedouble* produces such a maximal completed graph. If we apply *dedouble* to each of the two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, we obtain two completed graphs with a total of $\frac{n_1}{2} + 1 + \frac{n_2}{2} + 1$ cycles, that is, one more cycle than for $\mathcal{G}_3$. Thus, if we apply *dedouble* to the graphs of $\mathcal{SG}$ obtained by amalgamating graphs in $\mathcal{PES}$, we obtain $sp(G)$ fewer cycles than if we apply the algorithm to each graph of $\mathcal{PES}$. Moreover, one fewer cycle is also obtained by amalgamating one graph pair of $\mathcal{POS}$ and one graph of $\mathcal{PES}$. As these are the only modifications to the original procedure of graph amalgamating that changes the number of cycles, $c(\mathcal{G}_\Gamma) = c(G) - sp(G) - \psi(G)$.

Moreover, also according to Corollary 8.3 and Lemma 8.4, if G does not contain any PESs, then *dedouble* produces a maximal completed graph $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ with $s(\mathcal{G}_\Gamma)$ SPs corresponding to the $s(G)$ local SPs of G and to the only existing POS, if

any. Moreover, the $sr(G)$ potential SPs not amalgamated are the only potential SPs that become real SPs. Therefore, $s(\mathcal{G}) = s(G) - sr(G)$.

We deduce that

$$T(G,H) = |A| - c(\mathcal{G}_\Gamma) + s(\mathcal{G}_\Gamma) + f(\mathcal{G}_\Gamma) = |A| - c(G) + sp(G) + \psi(G) + s(G) + sr(G) + f(G).$$

Suppose $T(G,H) > T(G)$. Then there is a completed graph $\mathcal{G}_{\Gamma'}(\mathbf{V}, A, \Gamma')$ containing $c(\mathcal{G}_{\Gamma'})$ cycles, $s(\mathcal{G}_{\Gamma'})$ SPs, and a value of $f(\mathcal{G}_{\Gamma'})$ such that (1) $c(\mathcal{G}_{\Gamma'}) - s(\mathcal{G}_{\Gamma'}) - f(\mathcal{G}_{\Gamma'}) > c(\mathcal{G}_\Gamma) - s(\mathcal{G}_\Gamma) - f(\mathcal{G}_\Gamma)$, i.e., $c(\mathcal{G}_{\Gamma'}) - c(\mathcal{G}_\Gamma) > (s(\mathcal{G}_{\Gamma'}) - s(\mathcal{G}_\Gamma)) + (f(\mathcal{G}_{\Gamma'}) - f(\mathcal{G}_\Gamma))$.

First, suppose that $\mathcal{G}_{\Gamma'}(\mathbf{V}, A, \Gamma')$ contains p fewer SPs than $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$. Suppose first that $p = 1$ and that this SP is a real local SP of G. Then, by Lemma 8.4, $\mathcal{G}_{\Gamma'}(\mathbf{V}, A, \Gamma')$ is a completed graph that is not maximal and contains at most $c(\mathcal{G}_\Gamma) - 2$ cycles. More generally, a construction that removes p real local SPs of G gives rise to a completed graph with at most $c - 2p$ cycles. Suppose now that $\mathcal{G}_{\Gamma'}(\mathbf{V}, A, \Gamma')$ has one less SP than $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$, but this SP is not a real local SP of G. That means that it corresponds to a potential SP transformed into a final SP. This occurs in configurations C2, C3, C5, C6', C7, and C8. In all cases, it is easy to show that at least two cycles would necessarily be removed if we remove such an SP. Therefore, (2) $c(\mathcal{G}_{\Gamma'}) \leq c(\mathcal{G}_\Gamma) - 2(s(\mathcal{G}_{\Gamma'}) - s(\mathcal{G}_{\Gamma'}))$.

We deduce from (1) and (2) that $s(\mathcal{G}_\Gamma) - s(\mathcal{G}_{\Gamma'}) < f(\mathcal{G}_\Gamma) - f(\mathcal{G}_{\Gamma'})$.

As $s(\mathcal{G}_\Gamma) - s(\mathcal{G}_{\Gamma'}) \geq 0$ and $f(\mathcal{G}_\Gamma) - f(\mathcal{G}_{\Gamma'}) \leq 2$, we should have $s(\mathcal{G}_\Gamma) - s(\mathcal{G}_{\Gamma'}) = 1$ and $f(\mathcal{G}_\Gamma) - f(\mathcal{G}_{\Gamma'}) = 2$. We can see, from the definition of f, that this configuration is impossible.

Suppose now that $\mathcal{G}_{\Gamma'}(\mathbf{V}, A, \Gamma')$ contains p more SPs than $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$. If these SPs that are in $\mathcal{G}_{\Gamma'}(\mathbf{V}, A, \Gamma')$ but not in $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ do not correspond to potential local SPs of G, then, from Lemma 8.2 and the fact that $f(\mathcal{G}_\Gamma) \leq 2$, the value of $-c(\mathcal{G}_\Gamma) + s(\mathcal{G}_\Gamma + f(\mathcal{G}_\Gamma)$ is not changed if we remove these SPs. Thus these SPs correspond necessarily to potential local SPs that are transformed into final SPs in $\mathcal{G}_{\Gamma'}(\mathbf{V}, A, \Gamma')$ but not in $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$. Necessarily, $p \geq 2$.

Suppose first $p = 2$. If these two SPs correspond to

- two POS, then amalgamating these two SPs gives rise to two fewer SPs and to the same number of cycles,
- two PES, then amalgamating these two SPs gives rise to two fewer SPs and one less cycle,
- one POS and one PES, then amalgamating these SPs gives rise to two fewer SPs and one less cycle.

More generally, a graph containing $2p$ more SPs contains at most p more cycles than $\mathcal{G}(\mathbf{V}, A, \Gamma)$. Therefore, (3) $c(\mathcal{G}_{\Gamma'}) - c(\mathcal{G}_\Gamma) \leq \frac{s(\mathcal{G}_{\Gamma'}) - s(\mathcal{G}_\Gamma)}{2}$.

We deduce from (1) and (3) that $s(\mathcal{G}_{\Gamma'}) - s(\mathcal{G}_\Gamma) < 2(f(\mathcal{G}_\Gamma) - f(\mathcal{G}_{\Gamma'}))$.

As $s(\mathcal{G}_{\Gamma'}) - s(\mathcal{G}_\Gamma) \geq 2$ and $f(\mathcal{G}_\Gamma) - f(\mathcal{G}_{\Gamma'}) \leq 2$, $s(\mathcal{G}_{\Gamma'}) - s(\mathcal{G}_\Gamma) = 2$ and $f(\mathcal{G}_\Gamma) - f(\mathcal{G}_{\Gamma'}) = 2$. That means that amalgamating the two potential SPs gives rise to an even-isolation, which is in contradiction with the properties of the amalgamating procedure. □

10. Genome halving with translocations and reversals.

10.1. The HP method. Given two genomes H_1 and H_2 with the same number of chromosomes, HP [18] determined the minimal number of reversals and translocations $RT(H_1, H_2)$ required to transform H_1 into H_2. Formula **HP1** (section 3) is a general description of the result. A more precise description requires a deeper consideration of the problem.

We will only sketch the HP procedure, which is rather complex. The first step in the comparison of two multichromosomal genomes through translocations and reversals is to reduce it to a problem of comparing two single chromosome genomes through reversals only. These latter genomes are constructed essentially by concatenating the individual chromosomes in the original genomes end-to-end in an arbitrary order. Additional dummy genes, called *caps*, must be appropriately inserted at the ends of the original chromosomes of both genomes. A translocation in an original genome becomes a reversal in the new one.

More precisely, let $H = C_1, \ldots, C_N$ be a genome of N chromosomes written in a particular order. An H *concatenate* is a genome $\tilde{H}$ with a single chromosome: $\tilde{H} = (s_1 C_1) \cdots (s_N C_N)$, where each s_i, $1 \leq i \leq N$, is in $\{-1, 1\}$. The *identity concatenate* of H is the H concatenate satisfying $s = (s_1, \ldots, s_N) = (1, \ldots, 1)$.

Let $O = \{O_0, \ldots, O_{2N-1}\}$ be a set of caps and $\hat{H}_1$ the genome obtained by adding one cap at each end of each chromosome of H_1. Any sequence of reversals/translocations transforming H_1 into H_2 induces a sequence of reversals transforming $\hat{H}_1$ into a genome $\hat{H}_2$, where $\hat{H}_2$ is a particular capping of H_2. We can prove that $RT(H_1, H_2) = \min_{\hat{H}_2 \in \hat{\mathcal{H}}} RT(\hat{H}_1, \hat{H}_2)$, where $\hat{\mathcal{H}}$ is the set of all possible cappings of H_2.

Let $\tilde{H}_1$ be the identity concatenate of $\hat{H}_1$. Let $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ be the graph defined as follows: $V = \{x^{s \in \{t,h\}},\ x \in \mathcal{B}\}$ and $\mathbf{V} = V \cup O$; A is the set of black edges connecting adjacent vertices in H_1 other than (x^t, x^h) for the same x; Γ_s is the set of gray edges connecting adjacent vertices in H_2. Denote by V_e the set of vertices of V located at the ends of H_2 chromosomes. Note that vertices of $V_e \cup O$ are not connected by gray edges in $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$. $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ is called the *semicompleted graph* associated to H_1 and H_2. It is a collection of cycles and paths. Paths are of three kinds: those ending with a vertex in O and another in V, called *good paths*, those ending with two vertices in O, called *bad paths*, and those ending with two vertices in V. Denote, respectively, $\mathcal{OV}$, $\mathcal{OO}$, and $\mathcal{VV}$ as these three path sets. We have $|\mathcal{OO}| = |\mathcal{VV}|$.

A gray edge in a cycle or a path of size > 1 is *oriented* if it links the vertices at the left ends of two black edges or at the right ends of two black edges, while an *unoriented gray edge* links two different sides of two black edges. A cycle or a path is *good* if it contains at least one oriented gray edge, and *bad* otherwise. A component is *good* if it has at least one good cycle or path and thus at least one oriented gray edge, and *bad* otherwise.

HP showed that a good component can be *solved*, i.e., transformed to a set of cycles (and paths) of size 1, by a series of proper reversals (reversals increasing the number of cycles; see section 3). However, bad components often require bad reversals. The set of bad components is subdivided into subsets, depending on the difficulty of solving them (i.e., transforming them into good components). This subdivision is explained below.

An edge of Γ_s is *intrachromosomal* if it connects two vertices both belonging to the same chromosome of H_1 and *interchromosomal* otherwise. A component of $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ is intrachromosomal if it contains only intrachromosomal edges, and interchromosomal otherwise. We say that the component U *separates* two components U' and U'' if any edge we tried to draw from a vertex of U' to one of U'' would cut a gray edge of U. A *knot* is an intrachromosomal bad component which does not separate any pair of bad components. Now, a *real knot* is a knot that contains only cycles (no paths), and a *semiknot* is a knot containing at least one path in $\mathcal{OV}$ and no path in $\mathcal{OO} \cup \mathcal{VV}$.

The underlying idea is that a bad component U that separates two bad components U' and U'' is automatically solved by solving U' and U'' and thus may just as well be considered to be a good one. On the other hand, a real knot requires bad reversals to be solved, while a semiknot can be transformed into a good component if paths are closed appropriately.

HP proved that the problem of sorting by reversals/translocations is reduced to a problem of finding an appropriate capping of H_2, that is, finding appropriate connections between vertices of V_e and vertices of O. Finally, they proved that the minimal number of reversals/translocations required to transform H_1 into H_2 is

$$\textbf{HP3}: \quad RT(H_1,H_2) = b(\mathcal{G}_s) - cp(\mathcal{G}_s) + bp(\mathcal{G}_s) + rr(\mathcal{G}_s) + \left\lceil \frac{s(\mathcal{G}_s) - gr(\mathcal{G}_s) + fr(\mathcal{G}_s)}{2} \right\rceil,$$

where $b(\mathcal{G}_s) = |A|$; $cp(\mathcal{G}_s)$ is the number of cycles and paths of $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$; $bp(\mathcal{G}_s)$ is the number of bad paths; $rr(\mathcal{G}_s)$ is the number of real knots obtained after closing paths of $\mathcal{OV}$ that are not included in semiknots; $s(\mathcal{G}_s)$ is the number of semiknots; and $gr(\mathcal{G}_s)$ and $fr(\mathcal{G}_s)$ take values 0 or 1, depending on the set of real knots and semiknots.

Returning to the problem of genome halving, we represent the genome G as H_1, i.e., by adding caps at the ends of the chromosomes, and by concatenating the resulting chromosomes. The partial graph $\mathcal{G}(\mathbf{V}, A)$ associated to G is thus represented on a single line instead of $2N$ lines. Algorithm *dedouble* can be applied to such a partial graph as well. The goal is to construct a semicompleted graph $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ that minimizes formula **HP3**. We first construct a *maximal semicompleted graph* that maximizes $cp(\mathcal{G}_s) - bp(\mathcal{G}_s)$.

10.2. Constructing a maximal semicompleted graph. Denote by $c(G)$ the number of cycles of a maximal completed graph of $\mathcal{G}(\mathbf{V}, A)$ obtained by *dedouble*. For any semicompleted graph $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$, denote by $c(\mathcal{G}_s)$ its number of cycles and by $p(\mathcal{G}_s)$ its total number of paths. Denote also $cc(\mathcal{G}_s) = cp(\mathcal{G}_s) - bp(\mathcal{G}_s)$.

LEMMA 10.1. *Let $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ be a maximal semicompleted graph of $\mathcal{G}(\mathbf{V}, A)$. Then $cc(\mathcal{G}_s) \leq c(G)$.*

Proof. As mentioned above, we have $|\mathcal{OO}| = |\mathcal{VV}|$. Let $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ be the graph obtained by closing good paths and by constructing cycles with remaining paths, each of these paths obtained by connecting a path of $\mathcal{OO}$ with a path of $\mathcal{VV}$. Such a graph is clearly a completed graph of $\mathcal{G}(\mathbf{V}, A)$ with $c(\mathcal{G}_s) - p(\mathcal{G}_s)$ cycles. As $c(G)$ is the number of cycles of a maximal completed graph, we have $c(\mathcal{G}_s) - p(\mathcal{G}_s) \leq c(G)$. ∎

We now construct a semicompleted graph $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ satisfying $cc(\mathcal{G}_s) = c(G)$. From Lemma 10.1, this graph must be maximal.

THEOREM 10.2. *Let $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ be the maximal completed graph obtained by applying* dedouble *to $\mathcal{G}(\mathbf{V}, A)$. Let Γ_s be the set of gray edges obtained from Γ by removing all edges adjacent to at least one vertex in O, and consider the semicompleted graph $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$. Then $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ is a maximal semicompleted graph of $\mathcal{G}(\mathbf{V}, A)$. Moreover, $cp(\Gamma_s) = c(G)$.*

Proof. According to the construction of natural and supernatural graphs, each supernatural graph contains 0, 2, or 4 vertices in O. Moreover, it is easy to see that each cycle of a maximal completed graph contains at most two vertices in O as if this is not satisfied, then the cycle could be subdivided into at least two cycles, and thus the completed graph could not be maximal.

Let $\mathcal{G}'(\mathbf{V}', A')$ be a supernatural graph. Let $\mathcal{G}_{\Gamma'}(\mathbf{V}', A', \Gamma')$ be this supernatural graph completed by *dedouble*, and let $\mathcal{G}'_s(\mathbf{V}', A', \Gamma'_s)$ be the semicompleted supernat-

ural graph obtained by removing from Γ' edges with at least one end in O. Let $c(\Gamma')$ be the number of cycles of $\mathcal{G}_{\Gamma'}(\mathbf{V}', A', \Gamma')$.

- Suppose that $\mathbf{V}'$ does not contain any vertex in O. In this case, no edge is removed from Γ' to form $\mathcal{G}'_s(\mathbf{V}', A', \Gamma'_s)$, and thus $c(\mathcal{G}'_s) - p(\mathcal{G}'_s) = c(\Gamma')$.
- Suppose that $\mathbf{V}'$ contains two vertices in O. Suppose that these two vertices are in two different cycles of $\mathcal{G}_{\Gamma'}(\mathbf{V}', A', \Gamma')$. Then removing the two gray edges connecting these two vertices transforms the two corresponding cycles into two good paths. Thus $c(\mathcal{G}'_s) - p(\mathcal{G}'_s) = c(\Gamma')$.
  Suppose now that both vertices are in the same cycle. Then removing the two gray edges connecting these two vertices transforms the cycle into two paths, and at most one of them is bad. (In this case, the second path is in $\mathcal{VV}$.) Thus $c(\mathcal{G}'_s) - p(\mathcal{G}'_s) \geq (c(\Gamma') - 1) + 2 - 1 = c(\Gamma')$.
- Suppose that $\mathbf{V}'$ contains four vertices in O. If these vertices are in four or three different cycles, then we prove by an argument similar to the previous case that $c(\mathcal{G}'_s) - p(\mathcal{G}'_s) \geq c(\Gamma')$.
  Otherwise, if these vertices are in two cycles, then each of these cycles contains two of the four vertices. In that case, removing the four gray edges adjacent to these four vertices transforms the two cycles into four paths, and at most two of them are bad. Thus $c(\mathcal{G}'_s) - p(\mathcal{G}'_s) \geq (c(\Gamma') - 2) + 4 - 2 = c(\Gamma')$.

In all cases, $c(\mathcal{G}'_s) - p(\mathcal{G}'_s) \geq c(\Gamma')$. We deduce that $c(\mathcal{G}_s) - p(\mathcal{G}_s) \geq c(G)$. As $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ is a maximal completed graph, from Lemma 10.1, $c(\mathcal{G}_s) - p(\mathcal{G}_s) = c(G)$ and $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ is a maximal semicompleted graph. □

We call *semidedouble* the algorithm obtained by incorporating, at the end of *dedouble*, the procedure that removes the gray edges adjacent to at least one vertex of O. From Theorem 10.2, *semidedouble* constructs a maximal semicompleted graph.

Remark 3. Let C be a cycle of a completed graph $\mathcal{G}(\mathbf{V}, A, \Gamma)$ containing two vertices O_1 and O_2 in O. Let C_1 and C_2 be the two paths obtained by removing the two gray edges adjacent to O_1 and O_2. Then one of the following situations is satisfied: 1. C_1 and C_2 are two paths in $\mathcal{OV}$; or 2. C_1 is a bad path (in $\mathcal{OO}$) and C_2 is in $\mathcal{VV}$.

The first situation occurs when O_1 and O_2 are separated by an odd number of vertices in C (to the right or to the left), and the second situation occurs when O_1 and O_2 are separated by an even number of vertices.

10.3. Knots. We now turn our attention to minimizing, in formula **HP3**, the expression $rr(\mathcal{G}_s) + \lceil \frac{s(\mathcal{G}_s) - gr(\mathcal{G}_s) + fr(\mathcal{G}_s)}{2} \rceil$. Denote by $RT(G)$ the minimal number of reversals/translocations required to transform G into a duplicated genome. We deduce the following corollary from Theorem 7.7, Corollary 8.3, and formula **HP3**.

COROLLARY 10.3. *If G does not contain any local SP, then $RT(G) = T(G) = |A| - c(G) = \frac{|A|}{2} - \gamma(G)$.*

Suppose now that G contains local SPs. Let S be a local SP, and let $\Pi = \{\pi_1, \ldots, \pi_p\}$ be the set of components of the maximal semicompleted graph obtained by *semidedouble*, containing the vertices of S. In order to consider the components which may form knots, we introduce another definition. Let $\mathcal{U} = \{u_1, \ldots, u_p\}$ be a subset of $\mathcal{B}$, and $\overline{\mathcal{U}} = \{\overline{u_1}, \ldots, \overline{u_p}\}$. We say that $\mathcal{U}$ is *unoriented* if genes u_i and $\overline{u_i}$ have either the same sign in G or opposite signs for all i. Otherwise, $\mathcal{U}$ is *oriented*. Let $\pi_i \in \Pi$, and let $\mathbf{V_i}$ be its vertex set. $\mathbf{V_i}$ is oriented if and only if the set of genes corresponding to the vertices in $\mathbf{V_i}$ is oriented.

LEMMA 10.4. *π_i is a good component if and only if $\mathbf{V_i}$ is oriented.*

Proof. π_i is good if and only if π_i contains at least one cycle with at least one

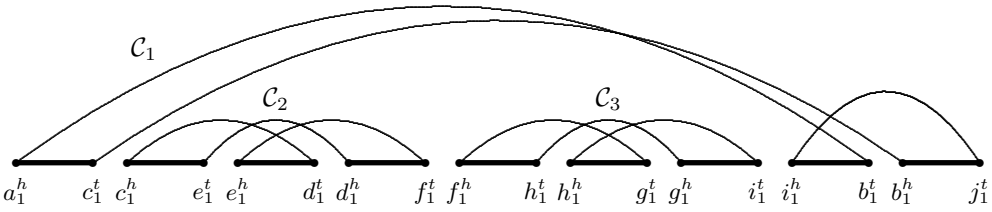


FIG. 10.1. *Inner SPs corresponding to* S_1.

oriented gray edge. Suppose that $\mathbf{V_i}$ is unoriented. We can suppose, without loss of generality, that all corresponding genes are signed +. All $\mathcal{C}$ cycles of π_i are such that left vertices are of form x^h and right edges are of form x^t (Definition 6.1). Moreover, there is no supernatural graph of $\mathcal{SO}$ obtained by amalgamating natural graphs containing vertices in $\mathbf{V_i}$. Therefore, a gray edge necessarily connects a left vertex to a right one in $\mathcal{C}$ cycles, and thus a vertex of form x^h to one of form x^t. From the definition of gray edge orientation, all gray edges of π_i are unoriented, and thus π_i is bad.

Conversely, if $\mathbf{V_i}$ is oriented, then we can assume, without loss of generality, that there exist two genes a and b such that a and b are adjacent and are both signed positively in S, but a and b do not have the same sign in $\overline{S}$. Algorithm *dedouble* constructs the gray edges (a^h, b^t) and $(\overline{a}^h, \overline{b}^t)$. As a and b do not have the same sign in $\overline{S}$, $\overline{a}^h$ and $\overline{b}^t$ are either both left ends or both right ends of black edges. Therefore, the gray edge $(\overline{a}^h, \overline{b}^t)$ is oriented. □

We say that a *real local SP is oriented* if the set of vertices in its component is oriented and is unoriented otherwise. Knots produced by *semidedouble* then correspond to the real minimal unoriented SPs, which we call *real minimal SPs*, and to at most one other SP, the maximal one, defined as follows.

DEFINITION 10.5. *Let* $S = x_1 \cdots x_n$ *be a local SP. The* outer SP *of* S *is the largest SP* S_e *contained in* S *satisfying the following three conditions:*

1. *The component* π_e *of* S_e *is bad;*
2. S_e *is not minimal, and the interval of* S_e *contains all the real minimal SPs of* S*;*
3. S_e *does not separate two real minimal SPs.*

A local SP S *is* maximal *if all the real minimal SPs of* G *are inner SPs of* S *and if there exists an outer SP of* S.

Example 6. Suppose that the genome G contains the local SP $S_1 = a_1\ c_1\ e_1\ d_1\ f_1\ h_1\ g_1\ i_1\ b_1\ j_1$, with complement $\overline{S_1} = a_2\ b_2\ c_2\ d_2\ e_2\ f_2\ g_2\ h_2\ i_2\ j_2$. The components of S_1 and the inner SPs of S_1 are depicted in Figure 10.1. The two components $\mathcal{C}_2$ and $\mathcal{C}_3$ correspond to components of the two minimal SPs of S. $\mathcal{C}_1$ is the component of S_1. It is bad and does not separate two minimal SPs. S_1 is thus an outer SP.

A *bad real SP* is a real SP which is either minimal or maximal. We denote by $\mathbf{brs}(\mathbf{G})$ the number of bad SPs of G.

As for semiknots, they are associated to potential SPs. To minimize them, we must minimize the number of potential SPs that become bad SPs of the final graph.

We already saw in section 9 how to solve POSs, provided that at least two of them exist. We can therefore assume that at most one POS exists. Suppose that one such POS S exists. In that case, we amalgamate the two odd size natural graphs G_1 and G_2 associated to S. Suppose that $G_1 = O_1 x_1 \cdots x_n O_1'$ and $G_2 = O_2 y_1 \cdots s\overline{x_1} \cdots y_n O_2'$,

where s is the sign of $\overline{x_1}$. If $s = +$, then we set $\overline{O_1} = O_2'$ and $\overline{O_1'} = O_2$. Otherwise ($s = -$), we set $\overline{O_1} = O_2$ and $\overline{O_1'} = O_2'$. With this construction, we ensure that the set of vertices $\{O_1^h, x_1, \ldots, x_n, O_1^{'t}\}$ is oriented.

Consider now the PESs. For every graph associated to a PES and containing two vertices O_1 and O_2 of O, *semidedouble* sets $\overline{O_1} = O_2$. We say that a PES S is unoriented if the set of vertices of S is unoriented and oriented otherwise. As oriented PESs do not give rise to any problem, we consider only, in the ensuing discussion, an unoriented PES S. S is a *minimal PES* if S is minimal. Moreover, an *outer PES* of S is defined in a similar way as for a real SP (Definition 10.5) by replacing "real minimal SPs" by "real minimal SPs and minimal PESs." We similarly define a *maximal PES* and a *bad PES*.

We denote by *BPES* a bad PES, b is the number of BPESs, and $\mathcal{BPES} = \{P_1, \ldots, P_b\}$ with, for every i, P_i as the set of graphs associated to the BPES i.

To minimize the number of semiknots, graphs of *BPES* are amalgamated with procedure *amalgamating-PES* described in section 9. Three configurations can arise after applying the procedure:

R1. There remains no nonamalgamated graph.
R2. There remains only one nonamalgamated graph, and it belongs to a P_i of size 2.
R3. There remains one $P_i \in \mathcal{BPES}$ with one or two nonamalgamated graphs. This happens if only one BPES exists or if b is odd and $\mathcal{BPES}$ contains only sets of size 1.

In the remainder of this section, $\mathcal{SG}$ is the set of completable graphs obtained by amalgamating the two graphs of a POS, if any, as described above, by using procedure *amalgamating-PES* for the graphs of $\mathcal{BPES}$, and by the usual way for the other natural graphs.

LEMMA 10.6. *Let $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ be the semicompleted graph obtained by* semidedouble. *Then $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ is a maximal semicompleted graph.*

Proof. Suppose that G does not contain any BPESs. The amalgamating procedure is then identical to that of section 5. Thus, from Theorem 10.2, *dedouble* constructs a maximal completed graph.

Suppose now that G contains BPESs. The only graphs of $\mathcal{SG}$ not corresponding to those of section 5 are those obtained by amalgamating graphs of $\mathcal{BPES}$. Let $\mathcal{G}_3(\mathbf{V}_3, A_3)$ be a graph of size n_3 obtained by amalgamating the two graphs $\mathcal{G}_1 = (\mathbf{V}_1, A_1)$ of size n_1 and $\mathcal{G}_2 = (\mathbf{V}_2, A_2)$ of size n_2 of $\mathcal{BPES}$. By arguments similar to those used in the proof of Theorem 9.3, we can see that *dedouble* gives rise to one less cycle when it is applied to a set of graphs containing $\mathcal{G}_3$, instead of a set of graphs containing the two supernatural graphs $\mathcal{G}_1$ and $\mathcal{G}_2$. More precisely, let c_{max} be the maximal number of cycles containing edges of $A_1 \cup A_2$ obtained when the two graphs $\mathcal{G}_1$, $\mathcal{G}_2$ are considered and c the number of cycles containing edges of A_3 obtained when $\mathcal{G}_3$ is considered. Then $c = c_{max} - 1$.

Let Γ_3' be the set of gray edges linking the vertices of $\mathbf{V}_3$ in a completed graph obtained by applying *dedouble* to a set of graphs containing $\mathcal{G}_3(\mathbf{V}_3, A_3)$. $\mathbf{V}_3$ has four vertices in O, denoted by the set O'. *Dedouble* constructs two cycles of size 1, each containing one of the vertices of O', and one cycle C of size > 1 containing the two remaining vertices of O'. Let $\mathcal{G}_{3,s}(\mathbf{V}_3, A_3, \Gamma_3)$ be the semicompleted graph of $\mathcal{G}_3(\mathbf{V}_3, A_3)$ obtained by *semidedouble*, that is, by removing from Γ_3' the edges adjacent to vertices of O. From Remark 3, vertices of O are either all left vertices or all right vertices. Therefore, removing from Γ_3' the edges adjacent to the vertices of O

transforms C into two good paths. The number of bad paths of $\mathcal{G}_{3,s}$ is then $bp_3 = 0$. Moreover, $cc(\mathcal{G}_{3,s}) = c_{max}$.

We deduce that $cc(\mathcal{G}_s) = c(G)$. $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ is thus a maximal semicompleted graph. □

A local SP that is either real or a BPES not solved by the procedure *amalgamating-PES* is called a *final SP*.

Notation 5. Consider the following parameters:

- $\mathbf{s(G)}$ is the number of BPESs that become semiknots. $s(G) = 0$ if configurations R1, R2 are encountered, and $s(G) = 1$ otherwise.
- $\mathbf{brs(G)}$ is the number of bad real SPs of G.
- $\mathbf{fr(G)}$ and $\mathbf{gr(G)}$ are defined like $fr(\mathcal{G}_s)$ and $gr(\mathcal{G}_s)$ [18]. They depend on the set of real knots and semiknots determined by the set of final SPs of G.

We require one more lemma.

LEMMA 10.7. *Suppose that G contains an unoriented local SP S. Let π be the component of S. Then any maximal completed graph must contain an unoriented component made up of the vertices of π.*

Proof. Suppose that G contains an unoriented real SP S. From Lemma 8.4, any maximal completed graph $\mathcal{G}(\mathbf{V}, A, \Gamma)$ contains an SP formed by S vertices. As S does not contain any vertex in O, any maximal semicompleted graph also contains an SP formed by S vertices. On the other hand, all supernatural graphs containing vertices of $S \cup \overline{S}$ are in $\mathcal{SE}$, and the corresponding completed supernatural graphs (in a maximal completed graph) give rise to at least one component. We want to show that each of these components contains exclusively unoriented gray edges.

As S is unoriented, we can assume that all genes corresponding to S vertices are signed positively and that all left vertices of S are of form x^h and all right vertices of form x^t. Let $\mathcal{G}_{sn}(\mathbf{V_{sn}}, A_{sn})$ be a supernatural graph containing vertices of S. Suppose that the corresponding completed supernatural graph $\mathcal{G}_{sn}(\mathbf{V_{sn}}, A_{sn}, \Gamma_{sn})$ contains an oriented edge. Such an edge necessarily links two left vertices or two right vertices. Arguments similar to those used in the proof of Lemma 6.3 show that $\mathcal{G}_{sn}(\mathbf{V_{sn}}, A_{sn}, \Gamma_{sn})$ contains at least one cycle less than a completed supernatural graph corresponding to a maximal completed graph. □

THEOREM 10.8. *Let $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ be the semicompleted graph produced by* semidouble. *Let H be the resulting duplicated genome. Then $cp(\mathcal{G}_s) - bp(\mathcal{G}_s) = c(G) = \frac{1}{2}|A| + \gamma(G)$, $rr(\mathcal{G}_s) = brs(G)$, $s(\mathcal{G}_s) = s(G)$, $fr(\mathcal{G}_s) = fr(G)$, $gr(\mathcal{G}_s) = gr(G)$, and*

$$RT(G,H) = \frac{1}{2}|A| - \gamma(G) + brs(G) + \left\lceil \frac{s(G) - gr(G) + fr(G)}{2} \right\rceil.$$

Moreover, $RT(G,H) = RT(G)$.

Proof. To simplify the notation, denote by cc, cp, bp, rr, s, gr, and fr the parameters corresponding to the graph $\mathcal{G}_s$.

From Lemma 10.6, $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$ is a maximal semicompleted graph, and $cp - bp = c(G)$. Now, we know that real knots correspond to bad real SPs, plus at most one maximal SP. Thus $rr = brs(G)$. As for semiknots, they correspond to bad components containing at least one good path and for which the corresponding interval does not contain any path in $\mathcal{OO} \cup \mathcal{VV}$. As POSs do not give rise to bad components, the only remaining nonamalgamated PES, if any, is the only SP giving rise to a bad component with vertices in O. As these vertices (in O) are either all left vertices or all right vertices, from Remark 3, removing gray edges adjacent to these vertices gives rise to only good cycles. One semiknot is then due to this nonamalgamated PES. Therefore, $s = s(G)$. We deduce

$$RT(G,H) = |A| - cp + bp + rr + \left\lceil \frac{s - gr + fr}{2} \right\rceil$$
$$= \frac{1}{2}|A| - \gamma(G) + brs(G) + \left\lceil \frac{s(G) - gr(G) + fr(G)}{2} \right\rceil.$$

Suppose $RT(G,H) > RT(G)$. This means that there exists a completed graph $\mathcal{G}_{s'}(\mathbf{V}, A, \Gamma'_s)$ with parameters cc', cp', bp', rr', s', gr', and fr' such that $cp' - bp' - rr' - \left\lceil \frac{s'-gr'+fr'}{2} \right\rceil > cp - bp - rr - \left\lceil \frac{s-gr+fr}{2} \right\rceil$, i.e.,

$$(1) \qquad cc' - cc > (rr' - rr) + \left(\left\lceil \frac{s' - gr' + fr'}{2} \right\rceil - \left\lceil \frac{s - gr + fr}{2} \right\rceil \right).$$

Suppose first that the completed graph $\mathcal{G}_{s'}(\mathbf{V}, A, \Gamma'_s)$ contains x fewer real knots than $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$. Suppose first that $x = 1$ and that it corresponds to a bad real SP. Then, from Lemma 10.7, $\mathcal{G}_{s'}(\mathbf{V}, A, \Gamma'_s)$ is a completed graph that is not maximal. More generally, a construction that removes x bad real SPs gives rise to a completed graph for which $cp' \leq c(G) - x$.

Suppose now that $\mathcal{G}_{s'}(\mathbf{V}, A, \Gamma'_s)$ contains one less semiknot than $\mathcal{G}_s(\mathbf{V}, A, \Gamma_s)$. This can occur in situations R2 or R3. However, removing such a semiknot, for example by constructing a good component, would also remove at least two cycles. Therefore,

$$(2) \qquad cp' - cp \leq (rr' - rr) + (s' - s).$$

We deduce from the above observations that

$$(\mathrm{I}) \qquad \left\lceil \frac{s' - gr' + fr'}{2} \right\rceil - \left\lceil \frac{s - gr + fr}{2} \right\rceil < s' - s.$$

Two possible situations occur:

1. $s' - gr' + fr'$ is even and $s - gr + fr$ is odd. In that case, inequality (I) induces $(s - s') < (-gr + fr) - (-gr' + fr') + 1$.
   But $s - s' \geq 0$ and $(-gr + fr) - (-gr' + fr') \leq 1$. This is due to the fact that if $fr = 1$, then $gr = 1$. (The same holds for fr' and gr'.)
   There are three possible cases: (a) $s - s' = 0$ and $(-gr + fr) - (-gr' + fr') = 0$; (b) $s - s' = 0$ and $(-gr + fr) - (-gr' + fr') = 1$; (c) $s - s' = 0$ and $(-gr + fr) - (-gr' + fr') = 1$.
   Cases (a) and (b) contradict the fact that $s' - gr' + fr'$ and $s - gr + fr$ are not both even or both odd.
2. All other situations for $s' - gr' + fr'$ and $s - gr + fr$ (other than $s' - gr' + fr'$ even and $s - gr + fr$ odd). In that case, inequality (I) induces $(s - s') < (-gr + fr) - (-gr' + fr')$. As $s - s' \geq 0$ and $(-gr + fr) - (-gr' + fr') \leq 1$, we should have $s - s' = 0$ and $(-gr + fr) - (-gr' + fr') = 1$.

Thus the only situation remaining is $s = s'$ and $(-gr + fr) = (-gr' + fr') + 1$. However, from the definitions of gr, fr, gr', and fr', this situation is impossible.

On the other hand, as the amalgamating procedure of $\mathcal{BPES}$ graphs preserves a maximal completed graph, a completed graph that contained more real knots or semiknots than $\mathcal{G}_s$ would not satisfy inequality (I). □

11. Genome halving with reversals.

11.1. The HP result. The problem of reconstructing a duplicated circular genome by reversals is a special case of the problem of reconstructing a duplicated multichromosomal genome by reversals and translocations. As the notion of endpoints is irrelevant for circular genomes, the distinction between a semicompleted graph and a completed graph is absent in this case. Let $\mathcal{G}(\mathbf{V}, A, \Gamma)$ be the completed graph obtained by *dedouble*. This graph can be decomposed into a set of alternating cycles (no paths). We define good and bad components in a similar way as for multichromosomal genomes (section 10), but by considering only cycles (no paths). Moreover, the concept of knots is here replaced by the concept of *hurdles*. Note that the concepts of "real hurdles" and "semihurdles" are irrelevant.

Let H_1 and H_2 be two single chromosome genomes, and let $\mathcal{G}_{12}$ be the breakpoint graph associated to H_1 and H_2. HP proved [17] that the minimal number of reversals required to transform H_1 to H_2 is

$$\textbf{HP4}: \quad R(H_1, H_2) = b(\mathcal{G}_{12}) - c(\mathcal{G}_{12}) + h(\mathcal{G}_{12}) + f(\mathcal{G}_{12}),$$

where $h(\mathcal{G}_{12})$ is the number of hurdles of $\mathcal{G}_{12}$ and $f(\mathcal{G}_{12})$ is a correction of size 0 or 1. In other words, in formula **HP1** (section 3), $m(\mathcal{G}_{12}) = h(\mathcal{G}_{12})$.

11.2. Maximizing the number of cycles. We denote by $R(G)$ the minimum number of reversals necessary to transform G into a duplicated genome. Denote by $c(G)$ the number of cycles of a maximal completed graph of $\mathcal{G}(\mathbf{V}, A)$. Theorem 6.6 gives an upper bound for $c(G)$. We would like to construct a completed graph with a number of cycles equal to this upper bound. This completed graph would then be maximal.

The method is almost identical to that described in section 7 for multichromosomal genomes. In particular, if we set $O = \emptyset$, then all the definitions and notation introduced in section 7 are valid for the circular genome case.

During the construction of gray edges, we still have to be careful not to create a circular fragment as long as unlinked vertices remain in the partially completed graph. In other words, the last step of the algorithm is the only one "closing" a fragment, eventually by constructing two gray edges of form $(x, \overline{x})$. Therefore, at each step except the last one (when there remain just four gray edges to be constructed to complete the graph), we have to construct possible pairs of gray edges, that is, pairs of gray edges that do not satisfy Properties I, II, and III (section 7).

In the case of circular genomes, if $\mathcal{SO}$ is not empty, then the set of "good" supernatural graphs, that is, the supernatural graphs of size $2n$ that can be completed by forming $n + 1$ cycles, contains one supernatural graph of $\mathcal{SO}$ (Lemma 6.5 and Theorem 6.6). However, constructing $n + 1$ cycles on a supernatural graph of $\mathcal{SO}$ creates a circular fragment. Therefore, to be able to construct a maximal number of cycles, we have to be careful to end up with a supernatural graph of $\mathcal{SO}$, if any.

The algorithm used in this case is also *dedouble*, with the slight difference described above. This algorithm constructs a maximal completed graph, that is, a completed graph with $c(\mathcal{G}_\Gamma) = \gamma + \frac{|A|}{2}$ cycles.

Example 7. Consider the genome $G = +a\ +b\ -c\ +b\ -d\ -e\ +a\ +c\ -d\ -e$. The decomposition of the partial graph into supernatural graphs is shown in Figure 11.1. We have $|A| = 10, \gamma = 3$, and thus $c = \gamma + \frac{A}{2} = 8$. Figure 11.1 depicts the completed graph produced by *dedouble*. The corresponding circular duplicated genome is

$$H = +c_1\ -b_1\ -a_1\ +e_2\ +d_2\ -d_1\ -e_1\ +a_2\ +b_2\ -c_2.$$

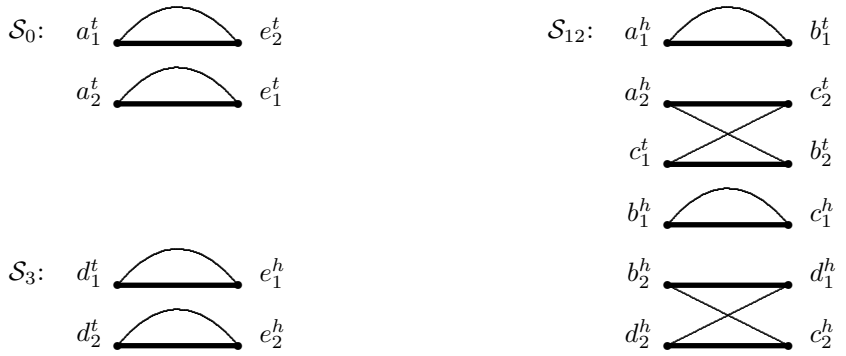


FIG. 11.1. *Completed graph* $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ *constructed by* dedouble.

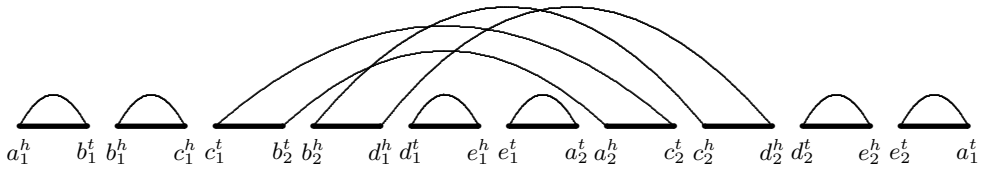


FIG. 11.2. *The completed graph constructed by* dedouble *for the genome* G *of Example* 7.

11.3. Hurdles. We now evaluate the number of hurdles contained in the maximal completed graph obtained by *dedouble*.

For circular genomes, the notion of a potential local SP is irrelevant, and only real local SPs remain. We saw, in section 8, how to modify *dedouble* so that, applied to a genome that does not contain any local SP, it gives rise to a completed graph containing no real SP.

The concepts of maximal, minimal, and bad SPs are defined as in section 10.3. Let $\mathbf{brs}(\mathbf{G})$ be the number of bad (real) SPs of G. Then, from Lemma 8.4 and Theorem 7.6, the completed graph $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ produced by *dedouble* contains exactly $brs(G)$ hurdles corresponding to these bad SPs. In addition, there may be at most two more special hurdles due to the special SPs defined in section 8.

Consider the parameter $f(G)$ which is 1 if the hurdles determined by the bad SPs of G form a fortress [17] and 0 otherwise. The next theorem is proved by arguments similar to those used for Theorem 10.8.

THEOREM 11.1. *Let* $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ *be the completed graph produced by* dedouble*, and let* H *be the resulting duplicated genome. Then*

$$\frac{|A|}{2} - \gamma(G) + brs(G) + f(G) \leq R(G, H) \leq \frac{|A|}{2} - \gamma(G) + brs(G) + f(G) + 2.$$

In addition,

$$\frac{|A|}{2} - \gamma(G) + brs(G) + f(G) \leq R(G) \leq \frac{|A|}{2} - \gamma(G) + brs(G) + f(G) + 2.$$

After Theorem 11.1, we have the following corollary.

COROLLARY 11.2. *Let* $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ *be the completed graph of* $\mathcal{G}(\mathbf{V}, A)$ *produced by* dedouble-circular. *If* $\mathcal{G}_\Gamma(\mathbf{V}, A, \Gamma)$ *does not contain any special hurdle, then*

$$R(G) = \frac{|A|}{2} - \gamma(G) + brs(G) + f(G).$$

Example 8. Consider genome G of Example 7 and the corresponding completed graph of Figure 7.5. Figure 11.2 gives a planar representation of this graph.

The number of cycles of this graph is $c(G) = 8$, $|A| = 10$, $brs(G) = 0$, and $f(G) = 0$. It does not contain any hurdles. Thus the minimum number of reversals necessary to transform genome G into a duplicated genome is $R(G) = 10-8+0+0 = 2$.

12. Analyzing the yeast genome. Wolfe and Shields [39] proposed that yeast is a degenerate tetraploid resulting from a genome duplication 10^8 years ago. They identified 55 duplicated regions, representing 50% of the genome.

12.1. Sorting by translocations. Applying our algorithm to the yeast genome data of Table 12.1, we obtain the perfect duplicated genome G_d represented in Table 12.2. The number of cycles of the corresponding completed graph $\mathcal{G}(V, A, \Gamma)$ is $c = 81$. Since G does not contain any local SPs, we can deduce that the minimal number of translocations required to transform G into G_d is

$$t = 2|\mathcal{B}| + |\mathcal{O}| - 2N - c = 142 - 16 - 81 = 45.$$

TABLE 12.1

Order of Wolfe and Shields' blocks on each of the 16 *chromosomes of the yeast genome. Signs indicate transcriptional orientation. In each chromosome, the* • *indicates the position of the centromere.*

I	:	+2 • −1
II	:	+4 • −3 − 7 +**8** − 5 + 6
III	:	+9 • −10 − 11
IV	:	+20 + 12 + 12 + 54 + 15 + 21 • −3 − 13 − 16 + 17 − 24 − 22 − 14
		−23 − 19 + 18 − 9
V	:	+28 • −25 − 27 − 4 − 26 − 13
VI	:	+55 • −36
VII	:	+36 + 25 + 26 + 32 +**6** − **33** + 5 • −30 − 34 − 31 − 29
VIII	:	+35 • −14 − 37 − 29 − 1
IX	:	+38 + 39 + 27 •
X	:	+10 + 40 + 41 • −28 − 42
XI	:	+42 + 40 + 43 + 35 • −41 − 52 − 38
XII	:	+53 • −53 − 31 − 55 − 16 −**18** − **17** − 45 − 30 − 15 − 44
XIII	:	+46 + 44 + 19 • −43 − 54 − 48 − 47 − 46
XIV	:	+49 + 20 + 37 + 50 + 39 • −11
XV	:	+49 + 21 • −22 − 52 − 50 − 23 − 45 − 51 − 47 − 2
XVI	:	+48 + 32 + 33 + 51 + 8 + 24 • −7 − 34

TABLE 12.2

Order of Wolfe and Shields' blocks on each of the 16 *chromosomes of the A solution for the ancestral genome. The present-day yeast genome can be obtained from this one by genome doubling followed by* 45 *translocations.*

1	:	+2 − 1
2	:	+46 + 47 + 48 + 54 + 43 + 35 − 41 − 40 − 42
3	:	+9 − 10 − 11
4	:	+44 + 15 + 21 − 22 − 14 − 23 − 19 + 18 + 16; +13 + 26
		+32 + 33 + 51 + 45 + 17 − 24 − 8 + 7 + 3 − 4
5	:	+55 − 36
6	:	+38 + 39 + 27 + 25 − 28
7	:	+29 + 37 + 50 + 52 − 53
8	:	+49 + 20 + 12 + 31 + 34 + 30 − 5 + 6

12.2. Sorting by reversals and translocations. As the yeast genome does not contain any real or potential local SPs, our method for sorting by reversals and translocations does not involve any reversals, so 45 translocations are still required.

13. An application on a circular genome. The mitochondrial genome of the liverwort plant *Marchantia polymorpha* is rather unusual in that many of its genes are manifested in two or three copies [30]. It is very unlikely that these arose from genome doubling, since this would not account for the numerous triplicates, nor is it consistent with comparative data on mitochondrial genomes. Nevertheless, it provides a convenient small example to test our method. A somewhat artificial map was extracted from the Genbank entry, deleting all singleton genes and one gene from each triplet. (The two genes furthest apart were saved from each triplet.) This led to a "rearranged duplicated genome" with 25 pairs of genes. A single supernatural graph in $\mathcal{SE}$ emerged from the analysis. This produced a minimum of 25 inversions, which is what one would expect from a random distribution of the duplicate genes on the genome. Any trace of genome duplication, were this even biologically plausible, has been obscured.

14. Conclusions. Calculating the HP formula for the edit distance between two genomes requires a rather intricate evaluation of the bicolored graph, including up to seven different structural parameters. In minimizing these formulae over the set of all (diploid) genomes, it is somewhat surprising that we can reconstruct an optimal ancestral genome exactly in all cases except the simplest reversals-only model. In the latter case, the uncertainty is not a deficiency of the algorithm but is due to ambiguity in how the doubled genome is constructed.

This work completes the major part of the program we undertook in [11]. In that article, we proposed a suite of "genome halving" problems and offered an algorithm for one of them in which a diploid genome is considered to be a set of genes partitioned among a number of subsets called chromosomes. The only operation is translocation considered as an exchange of subsets between two chromosomes. For the reconstruction problem in that context, in all likelihood NP-hard, we offered an effective heuristic which functions well on trial data. The present work shows that by adding gene order and transcription direction (strandedness, sign, polarity) to chromosome structure and adding the reversal operation, exact linear algorithms are possible. (Gene order alone, without transcription direction, would likely not suffice to permit polynomial-time exact algorithms; cf. [7].)

An additional level of structure to increase the realism of the model would be to incorporate a *centromere* on each chromosome. The centromere can occur anywhere in the linear order of genes, the centromeres are structurally indistinguishable from each other (for our purposes), and there is normally exactly one centromere per chromosome. This condition excludes some translocations, namely, those which result in one chromosome with two centromeres and one with none. The algorithms we have developed for the reconstruction of doubled genomes are not easily adaptable in this context. It should be noted that the condition on single centromeres is occasionally violated in nature, as, for example, with fissions and fusions, so that ideally the model should be complicated to allow for centromere creation and disappearance. As a final note, this work, together with [12] on hybridization and [9] on segment duplication, represents the use of computational biology techniques first developed for comparative genomics, as tools for the internal reconstruction of the evolutionary history of a single genome.

Acknowledgments. The authors are Scholar and Fellow of the Evolutionary Biology Program of the Canadian Institute for Advanced Research.

REFERENCES

[1] S. Ahn and S. D. Tanksley, *Comparative linkage maps of rice and maize genomes*, Proc. Natl. Acad. Sci. USA, 90 (1993), pp. 7980–7984.

[2] D. J. Amor and K. H. Choo, *Links neocentromeres: Role in human disease, evolution, and centromere study*, Amer. J. Human Genetics, 71 (2002), pp. 695–714.

[3] Arabidopsis genome initiative, *Analysis of the genome sequence of the flowering plant Arabidopsis thaliana*, Nature, 408 (2000), pp. 796–815.

[4] N. B. Atkin and S. Ohno, *DNA values of four primitive chordates*, Chromosoma, 23 (1967), pp. 10–13.

[5] D. A. Bader, B. M. E. Moret, and M. Yan, *A linear-time algorithm for computing inversion distances between signed permutations with an experimental study*, J. Comput. Biol., 8 (2001), pp. 483–491.

[6] A. Bergeron, *A very elementary presentation of the Hannenhalli-Pevzner theory*, in Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 2089, A. Amir and G. M. Landau, eds., Springer-Verlag, New York, 2001, pp. 106–117.

[7] A. Caprara, *Sorting by reversals is difficult*, in Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB), ACM, New York, 1997, pp. 75–83.

[8] D. Durand, *Vertebrate evolution: Doubling and shuffling with a full deck*, Trends in Genetics, 19 (2003), pp. 2–5.

[9] N. El-Mabrouk, *Reconstructing an ancestral genome using minimum segments duplications and reversals*, J. Comput. System Sci., Special Issue on Computational Molecular Biology, 65 (2002), pp. 442–464.

[10] N. El-Mabrouk, B. Bryant, and D. Sankoff, *Reconstructing the pre-doubling genome*, in Proceedings of the Third Annual International Conference on Computational Molecular Biology (RECOMB), ACM, New York, 1999, pp. 154–163.

[11] N. El-Mabrouk, J. H. Nadeau, and D. Sankoff, *Genome halving*, in Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 1448, M. Farach, ed., Springer-Verlag, New York, 1998, pp. 235–250.

[12] N. El-Mabrouk and D. Sankoff, *Hybridization and genome rearrangement*, in Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 1645, M. Crochemore and M. Paterson, eds., Springer-Verlag, New York, 1999, pp. 78–87.

[13] R. Friedman and A. L. Hughes, *Pattern and timing of gene duplication in animal genomes*, Genome Res., 11 (2001), pp. 1842–1847.

[14] K. J. Fryxell, *The coevolution of gene family trees*, Trends in Genetics, 12 (1996), pp. 364–369.

[15] B. S. Gaut and J. F. Doebley, *DNA sequence evidence for the segmental allotetraploid origin of maize*, Proc. Natl. Acad. Sci. USA, 94 (1997), pp. 6809–6814.

[16] S. Hannenhalli, *Polynomial-time algorithm for computing translocation distance between genomes*, in Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 937, Z. Galil and E. Ukkonen, eds., Springer-Verlag, New York, 1995, pp. 162–176.

[17] S. Hannenhalli and P. A. Pevzner, *Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals)*, in Proceedings of the 27th Annual ACM–SIAM Symposium on Theory of Computing, ACM, New York, SIAM, Philadelphia, 1995, pp. 178–189.

[18] S. Hannenhalli and P. A. Pevzner, *Transforming men into mice (polynomial algorithm for genomic distance problem)*, in Proceedings of the 36th IEEE Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1995, pp. 581–592.

[19] M. Herdman, *The evolution of bacterial genomes*, in The Evolution of Genome Size, T. Cavalier-Smith, ed., John Wiley and Sons, New York, 1985, pp. 37–68.

[20] R. Hinegardner, *Evolution of cellular DNS content in teleost fishes*, American Naturalist, 102 (1968), pp. 517–523.

[21] A. L. Hughes, *Phylogenies of developmentally important proteins do not support the hypothesis of two rounds of genome duplication early in vertebrate history*, J. Molecular Evolution, 48 (1999), pp. 565–576.

[22] H. Kaplan, R. Shamir, and R. E. Tarjan, *A faster and simpler algorithm for sorting signed permutations by reversals*, SIAM J. Comput., 29 (1999), pp. 880–892.

[23] T. Kunisawa, *Identification and chromosomal distribution of DNA sequence segments conserved since divergence of Escherichia coli and Bacillus subtilis*, J. Molecular Evolution, 40 (1995), pp. 585–593.

[24] M. Lynch and J. S. Conery, *The evolutionary fate and consequences of duplicate genes*, Science, 290 (2000), pp. 1151–1155.

[25] A. McLysaght, K. Hokamp, and K. H. Wolfe, *Extensive genomic duplication during early chordate evolution*, Nature Genetics, 31 (2002), pp. 200–204.

[26] G. Moore, K. M. Devos, Z. Wang, and M. D. Gale, *Grasses, line up and form a circle*, Current Biology, 5 (1995), pp. 737–739.

[27] F. Muller, V. Bernard, and H. Tobler, *Chromatin diminution in nematodes*, Bioessays, 18 (1996), pp. 133–138.

[28] J. H. Nadeau, *Genome duplication and comparative mapping*, in Advanced Techniques in Chromosome Research, K. T. Adolph, ed., Marcel Dekker, New York, 1991, pp. 269–296.

[29] J. H. Nadeau and D. Sankoff, *Comparable rates of gene loss and functional divergence after genome duplications early in vertebrate evolution*, Genetics, 147 (1997), pp. 1259–1266.

[30] K. Oda, K. Yamato, E. Ohta, Y. Nakamura, M. Takemura, N. Nozato, T. Kohchi, Y. Ogura, T. Kanegae, K. Akashi, and K. Ohyama, *Gene organization deduced from the complete sequence of liverwort Marchantia polymorpha mitochondrial DNA. A primitive form of plant mitochondrial genome*, J. Molecular Biol., 223 (1992), pp. 1–7.

[31] S. Ohno, U. Wolf, and N. B. Atkin, *Evolution from fish to mammals by gene duplication*, Hereditas, 59 (1968), pp. 169–187.

[32] A. H. Paterson, T. H. Lan, K. P. Reischmann, C. Chang, Y. R. Lin, S. C. Liu, M. D. Burow, S. P. Kowalski, C. S. Katsar, T. A. DelMonte, K. A. Feldmann, K. F. Schertz, and J. F. Wendel, *Toward a unified genetic map of higher plants, transcending the monocot-dicot divergence*, Nature Genetics, 14 (1996), pp. 380–382.

[33] J. H. Postlethwait, Y. L. Yan, M. A. Gates, S. Horne, A. Amores, A. Brownlie, A. Donovan, E. S. Egan, A. Force, Z. Gong, C. Goutel, A. Fritz, R. Kelsh, E. Knapik, E. Liao, B. Paw, D. Ransom, A. Singer, T. Thomson, T. S. Abduljabbar, P. Yelick, D. Beier, J. S. Joly, D. Larhammar, F. Rosa, M. Westerfield, L. I. Zon, and W. S. Talbot, *Vertebrate genome evolution and the zebrafish gene map*, Nature Genetics, 18 (1998), pp. 345–349.

[34] J. A. Scheffler, A. G. Sharpe, H. Schmidt, P. Sperling, I. A. P. Parkin, W. Lühs, D. J. Lydiate, and E. Heinz, *Desaturase multigene families of Brassica napus arose through genome duplication*, Theoretical and Applied Genetics, 94 (1997), pp. 583–591.

[35] C. Seoighe and K. H. Wolfe, *Extent of genomic rearrangement after genome duplication in yeast*, Proc. Natl. Acad. Sci. USA, 95 (1998), pp. 4447–4452.

[36] R. C. Shoemaker, K. Polzin, J. Labate, J. Specht, E. C. Brummer, T. Olson, N. Young, V. Concibido, J. Wilcox, J. P. Tamulonis, G. Kochert, and H. R. Boerma, *Genome duplication in soybean (Glycine subgenus soja)*, Genetics, 144 (1996), pp. 329–228.

[37] A. C. Siepel, *An algorithm to find all sorting reversals*, in Proceedings of the 6th Annual International Conference on Computational Molecular Biology (RECOMB), ACM, New York, 2002, pp. 281–290.

[38] L. Skrabanek and K. H. Wolfe, *Eukaryote genome duplication, where's the evidence?*, Current Opinion in Genetics and Development, 8 (1998), pp. 694–700.

[39] K. H. Wolfe and D. C. Shields, *Molecular evidence for an ancient duplication of the entire yeast genome*, Nature, 387 (1997), pp. 708–713.

[40] R. H. Xu, J. Kim, M. Taira, J. J. Lin, C. H. Zhang, D. Sredni, T. Evans, and H. F. Kung, *Differential regulation of neurogenesis by the two Xenopus GATA-1 genes*, Molecular and Cellular Biology, 17 (1997), pp. 436–443.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 793–815

EXPECTED-CASE COMPLEXITY OF APPROXIMATE NEAREST NEIGHBOR SEARCHING*

SUNIL ARYA† AND HO-YAM ADDY FU†

Abstract. Most research in algorithms for geometric query problems has focused on their worst-case performance. However, when information on the query distribution is available, the alternative paradigm of designing and analyzing algorithms from the perspective of expected-case performance appears more attractive. We study the approximate nearest neighbor problem from this perspective.

As a first step in this direction, we assume that the query points are sampled uniformly from a hypercube that encloses all the data points; however, we make no assumption on the distribution of the data points. We show that with a simple partition tree, called the sliding-midpoint tree, it is possible to achieve linear space and logarithmic query time in the expected case; in contrast, the data structures known to achieve linear space and logarithmic query time in the worst case are complex, and algorithms on them run more slowly in practice. Moreover, we prove that the sliding-midpoint tree achieves optimal expected query time in a certain class of algorithms.

Key words. nearest neighbor searching, approximation, expected-case analysis, priority search, sliding-midpoint tree

AMS subject classifications. 68P05, 68Q25, 68U05

PII. S0097539799366340

1. Introduction.

The main focus in the design of data structures and algorithms for geometric query problems has been to obtain optimal worst-case query time. However, in many applications, the average time for answering a query is more important than the worst-case time. Thus, when we have information on the query distribution, we believe it is prudent to incorporate it in the algorithm design with a view to minimize the *expected* query time and to provide simpler data structures. We study the approximate nearest neighbor problem from this novel perspective.

Nearest neighbor searching is a fundamental problem in computational geometry with applications in numerous areas such as pattern recognition [11], data compression [17], information retrieval [10], and multimedia databases [15]. Since the problem is very difficult to solve exactly in high dimensions, researchers have investigated the *approximate* nearest neighbor problem. Consider a set S of n data points in R^d and a query point $q \in R^d$. Given an error bound $\epsilon > 0$, we say that a point $p \in S$ is a $(1+\epsilon)$-*approximate nearest neighbor* of q if $dist(p,q) \leq (1+\epsilon)dist(p^*,q)$, where p^* is the true nearest neighbor of q. This problem has been extensively studied from the worst-case perspective [2, 3, 5, 6, 7, 8, 9, 12, 18, 19, 21].

For our expected-case study, we consider that the set S of n data points is contained within the unit hypercube $U = [0,1]^d$ and assume that the query points are sampled from the uniform distribution in U. Note that we make no assumption on the distribution of the data points. While the assumption of uniformly distributed query points is admittedly simplistic, we think it is a natural first step toward the design of algorithms for more general query distributions. We investigate a general approach for

*Received by the editors December 13, 1999; accepted for publication (in revised form) February 3, 2003; published electronically April 23, 2003. This research was supported in part by RGC CERG HKUST736/96E and RGC DAG96/97.EG40. A preliminary version of this paper appeared in the *Proceedings of the 11th Annual ACM–SIAM Symposium on Discrete Algorithms*, 2000, pp. 379–388.
http://www.siam.org/journals/sicomp/32-3/36634.html

†Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (arya@cs.ust.hk, csaddy@cs.ust.hk).

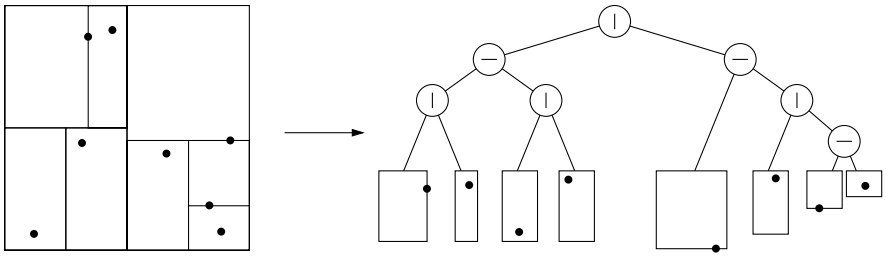


FIG. 1. *Sliding-midpoint tree.*

finding the approximate nearest neighbor devised by Arya et al. [4, 6], called *priority search*. This approach can be used in conjunction with any partition tree for S and is based on efficiently enumerating the leaves of the tree in order of increasing distance from the query point until a certain termination condition is satisfied. (Section 3 describes this method and presents its important features.) In this study, we consider the question of how to construct the partition tree so as to minimize the expected query time.

Our main results are for a partition tree based on a very simple splitting method called the *sliding-midpoint method*, introduced by Mount and Arya [24]. As in the standard kd-tree [16], cells are recursively subdivided using hyperplanes that are orthogonal to the coordinate-axes. More precisely, we first place a hyperplane orthogonal to the longest side of the cell and at its middle. We let this hyperplane be the splitting plane if there are data points located on both sides; otherwise, we slide the hyperplane toward the side that contains all the data points until it just touches a point. The point that touches the splitting plane is assigned to the side that is originally empty. (See Figure 1.) The space used by this tree is $O(n)$ (since no empty cells are created), and it can be easily constructed in $O(n \log n)$ time using well-known techniques [6]. Although its worst-case query time can be as bad as $\Omega(n)$, Maneewongvatana and Mount [22] have studied it empirically and observed that it performs well in practice. We present two results pertaining to its expected query time that provide some theoretical justification for its good practical performance.

First, we prove that the expected query time for this tree is $O((1/\epsilon)^d \log n)$, irrespective of the distribution of data points. We note that Arya et al. [6] and, more recently, Duncan, Goodrich, and Kobourov [12] have proposed partition trees for which priority search achieves logarithmic worst-case query time. However, they both use considerably more complex ways of subdividing a cell. (In addition to axis-orthogonal splits, Arya et al. allow a *shrink* operation, which can generate a cell that is the set-theoretic difference of two rectangles. Duncan, Goodrich, and Kobourov restrict themselves to splits with a hyperplane; however, the hyperplane is not necessarily axis-orthogonal.) Although both these partition trees provide optimal worst-case query time, the features introduced to ensure this property hurt their practical performance.

Our second result, which is of greater significance, shows that the sliding-midpoint tree is, in fact, optimal in a certain sense. Consider the class of algorithms obtained by running priority search on partition trees, where the splits are made by hyperplanes at arbitrary orientations (that is, not necessarily axis-orthogonal). We prove that, for any given data distribution, priority search on the sliding-midpoint tree achieves the minimum expected query time (up to a constant factor depending on d and ϵ)

over all such algorithms. Our proof also implies that priority search on the sliding-midpoint tree attains lower expected query time compared to any algorithm that can be modeled as an algebraic decision tree using linear tests (that is, involving the evaluation of a polynomial of degree one).

The paper is organized as follows. In section 2, we describe our notation. Section 3 reviews the priority search approach for finding the approximate nearest neighbor. Section 4 presents a general framework for the expected-case analysis of partition trees. In section 5, we give a simple proof that the expected query time using the sliding-midpoint tree is $O(\log n)$, and, in section 6, we establish the optimality of this tree. Finally, we present our experimental results in section 7.

2. Conventions. Let U denote the unit hypercube $[0,1]^d$ in d dimensions. We assume that the set S of n data points has been scaled and translated to lie within U. To avoid confusion, we always use *data point* to refer to a point in the data set S, and we use *point* to refer to any point in space. We assume that the dimension d and the error bound ϵ are fixed constants, independent of the number of data points.

We will assume that distances are measured in the Euclidean metric (although our results can be easily generalized to any Minkowski metric). We define the distance between a point p and a region z to be the minimum distance between p and any point in z.

Throughout, the word *rectangle* will denote a d-dimensional axis-parallel hyper-rectangle. We define the *size* of a rectangle to be the length of its longest side. For any region z, we let v_z denote its volume (area in two dimensions and length in one dimension). If z is the set-theoretic difference of two rectangles, one enclosed within the other, we denote the *outer rectangle* and *inner rectangle* by z_O and z_I, respectively, and we define its *size* to be the size of its outer rectangle.

The data structures we consider are based on partition trees that represent a hierarchical decomposition of U. We recall some basic facts about partition trees. Each node of the partition tree is associated with a region of space, called a *cell*, and the set of data points that lie in this cell. The cell associated with any node is partitioned into disjoint cells and associated with the children of the node. We assume that the cell associated with the root of the partition tree is the unit hypercube U. The leaves of the tree contain at most a constant number of data points, called the *bucket size* (henceforth assumed to be ≤ 1 for simplicity).

Given a partition tree T, let $\mathcal{I}_T, \mathcal{L}_T$, and $\mathcal{N}_T$ denote the set of its internal nodes, leaf nodes, and all nodes, respectively. We let $\mathcal{Z}_T$ denote the subdivision of U induced by the leaf nodes of T. Let x be a node in T. We let C_x denote the cell associated with x and v_x denote the volume of C_x. If C_x is a rectangle or the difference of two rectangles, we use s_x to denote the size of C_x. We will often use R_x in place of C_x if C_x is a rectangle. Finally, we use ℓ_x to denote the level of x, that is, the length of the path from the root to x. Note that the root is at level 0.

3. Background. We briefly review some features of the priority search approach for approximate nearest neighbor searching, proposed by Arya and Mount [4] and Arya et al. [6].

3.1. Query algorithm. The algorithm is based on visiting leaf cells in order of increasing distance from the query point q. For each leaf cell visited, the algorithm computes the distance between q and the data point stored with the leaf and updates p, the closest neighbor to q found so far, and r_p, the distance to it. The algorithm terminates when the distance between q and the leaf cell exceeds $r_p/(1+\epsilon)$, returning

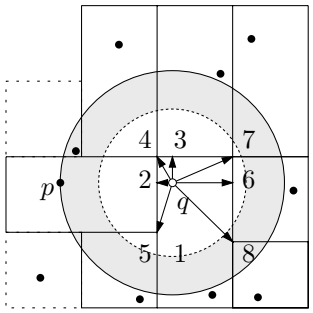


FIG. 2. *Algorithm overview.*

p as the answer. For example, in Figure 2, the leaf cells have been labeled in order of increasing distance from the query point q. The algorithm terminates on visiting cell 8, since its distance from the query point exceeds $r_p/(1+\epsilon)$ (the radius of the dotted circle shown).

To see that the algorithm works correctly for any partition tree, let B denote the ball of radius $r_p/(1+\epsilon)$ centered at q. Since the algorithm has already seen all leaf cells overlapping B, it is clear that B contains no data point. Thus point p is a $(1+\epsilon)$-approximate nearest neighbor of q.

It remains to describe the method used to enumerate the leaf cells in order of increasing distance from the query point. The algorithm maintains a priority queue of nodes, where the priority of a node is inversely related to the distance between the query point and the corresponding cell. Initially, the root of the tree is inserted into the queue. Then the following procedure is repeatedly carried out. First, the node with the highest priority is extracted from the queue, that is, the node closest to the query point. Then the algorithm descends the subtree associated with this node until reaching the leaf closest to the query point. For each internal node visited, the distance between the query point and the children of the node is computed. The algorithm descends to the child that is closer to the query point, while the other child is inserted into the queue. When the algorithm reaches the leaf, it processes the data point associated with it. The correctness of the algorithm is based on the invariant that the set of leaves descended from the nodes in the priority queue is disjoint and their union is the set of all unvisited leaves.

While priority search can be applied on any partition tree, it is especially efficient on trees that use only axis-orthogonal splits. The reason is that, for each internal node visited, the algorithm needs to compute the distance between the query point and the children of the node. If we do not restrict ourselves to axis-orthogonal splits, this computation can take time proportional to the complexity of the cell. In contrast, for a node that is partitioned by an axis-orthogonal hyperplane, this computation can be done in $O(1)$ time *independent* of dimension d, using a technique called *incremental distance computation*. From a practical perspective, the difference is significant especially in high dimensions. Intuitively, this technique works because the cells for the children differ from the parent's cell in only one of the d dimensions. For details, we refer the reader to Arya and Mount [4].

3.2. Query time. The analysis of the query time given in [6] employs the following two lemmas, which will also be useful for us. Lemma 3.1 says that all the leaf cells visited, except possibly the last one, are large relative to their distance from the

query point. Lemma 3.2 relates the query time to the number of internal nodes and leaf nodes visited. Their proofs follow from the discussion in [6]; we repeat them here for the sake of completeness.

LEMMA 3.1 (see [6]). *Consider a partition tree T built using axis-orthogonal splits, such that each leaf cell contains a data point. Suppose that we run priority search on T with query point q and error bound ϵ. Then the size of any leaf cell x that does not cause the algorithm to terminate is at least $r\epsilon/\sqrt{d}$, where r is the distance from q to x.*

Proof. Suppose that the size of x is less than $r\epsilon/\sqrt{d}$. Clearly, the diameter of x is then less than $r\epsilon$, and so it contains a data point at distance less than $r(1+\epsilon)$ from q. This implies that x must satisfy the termination condition, which is a contradiction. □

LEMMA 3.2 (see [4, 6]). *Consider a partition tree T built using axis-orthogonal splits. Suppose that we run priority search on T with query point q and error bound ϵ. Let I and L be the number of internal nodes and leaf nodes, respectively, visited by the algorithm. Then the query time is $O(I + Ld + L\log I)$, where the constant factor in the O-notation is independent of d and ϵ.*

Proof. On visiting an internal node, the algorithm first updates the distance to its two children; since the children are created by an axis-orthogonal split, this can be done in $O(1)$ time using incremental distance computation. Then it inserts the farther child into the priority queue; using Fibonacci heaps, the amortized time for each insertion is $O(1)$. Finally, it descends to the closer child, which takes $O(1)$ time.

On visiting a leaf node, it computes the distance between q and the data point stored with the leaf, which takes $O(d)$ time. It then extracts the closest node from the queue. Since each internal node visited inserts one child into the queue, the size of the queue is at most I. Thus it takes $O(\log I)$ time to extract the closest node. The total query time is therefore $O(I + L(d + \log I)) = O(I + Ld + L\log I)$. □

4. Expected-case analysis: General framework. In this section, we set up the basic framework for analyzing the expected query time of priority search on a partition tree. For the sake of simplicity, we assume that the partition tree is built using axis-orthogonal splitting planes. Before presenting the analysis, we first need to make a small but important modification to priority search.

4.1. Modified priority search. We give some intuition for why this modification is needed. Recall from section 3 that priority search is given a chance to terminate the search only at leaf nodes. Further, it follows from Lemma 3.1 that it visits at most one leaf cell of size less than $r\epsilon/\sqrt{d}$, where r is the distance from the query point to the leaf cell. However, while descending the tree to this small leaf cell, the algorithm may visit a large number of small internal nodes. This is undesirable as it increases the query time and prevents us from proving good bounds on the expected query time. Thus the idea of the modification is to allow the search to be terminated at internal nodes as well to ensure that at most one node (leaf or internal) of size less than $r\epsilon/\sqrt{d}$ is visited.

The details of the modification are as follows. With each internal node x, we store its size s_x and a pointer to any data point p_x inside R_x. (Asymptotically, this does not increase the space or preprocessing requirements.) The search algorithm works in exactly the same way if a leaf node is visited. If an internal node x is visited, then let r denote its distance from the query point q. If $s_x \geq r\epsilon/\sqrt{d}$, the algorithm works just as before. However, if $s_x < r\epsilon/\sqrt{d}$, the algorithm computes the distance from q to the associated data point p_x, updates the closest point seen so far, and terminates.

It is easy to see that the modified algorithm remains correct. Obviously, if the algorithm terminates at a leaf node, then it behaves exactly as the unmodified algorithm and is therefore correct. If it terminates at an internal node x at a distance r from q, then the diameter of R_x is at most $r\epsilon$, and so the associated data point p_x is at a distance at most $r(1+\epsilon)$ from q. Note that the algorithm has already seen any data point that lies in the ball of radius r centered at q. Clearly, if this ball contains no data point, then p_x is a $(1+\epsilon)$-approximate nearest neighbor. Otherwise, the algorithm returns the nearest neighbor of q. In either case, the algorithm returns the correct answer.

For the rest of this paper, we shall assume that priority search has been modified as indicated above. Note that Lemmas 3.1 and 3.2 continue to hold after this modification.

4.2. Upper bound on expected number of nodes visited. Recall that S is a set of n data points in U and the query point is sampled from the uniform distribution in U. In Lemma 4.1, we establish upper bounds on the expected number of internal nodes and leaf nodes visited by priority search, which depend only on the sizes of the rectangles associated with the nodes of the partition tree. Together with Lemma 3.2, these upper bounds facilitate the expected-case analysis of partition trees formed by different splitting methods. In sections 5 and 6, we will apply them to the sliding-midpoint tree.

LEMMA 4.1. *Let d and ϵ be any fixed constants. Let S be a set of n data points in U. Let T be a partition tree for S built using axis-orthogonal splits, such that each leaf cell contains a data point. Assume that the query point is sampled from the uniform distribution in U. Then the expected number of internal nodes and leaf nodes visited by priority search is at most $1+(1+2\sqrt{d}/\epsilon)^d \sum_{x\in\mathcal{I}_T} s_x^d$ and $1+(1+2\sqrt{d}/\epsilon)^d \sum_{x\in\mathcal{L}_T} s_x^d$, respectively.*

Proof. We will prove only the bound on the expected number of internal nodes visited, since the proof for leaf nodes is similar. Let q be a query point. Let x be any node visited that does not cause the algorithm to terminate, and let r be the distance between q and R_x. We claim that $s_x \geq r\epsilon/\sqrt{d}$. If x is an internal node, this follows in view of the modifications made to priority search, and if x is a leaf node, this follows from Lemma 3.1. Thus $r \leq s_x\sqrt{d}/\epsilon$. That is, if a node x is visited and does not cause the algorithm to terminate, then the query point q must be at a distance at most $s_x\sqrt{d}/\epsilon$ from R_x.

With each internal node x, associate a random variable V_x that takes the value 1 if node x is visited and does not cause the algorithm to terminate; otherwise, it takes the value 0. Let I denote the number of internal nodes visited by priority search. Clearly $I \leq 1+\sum_{x\in\mathcal{I}_T} V_x$. Here we have added 1 to take into account the last internal node visited, which may have caused the algorithm to terminate. By linearity of expectation,

$$E[I] \leq 1 + \sum_{x\in\mathcal{I}_T} E[V_x]. \tag{1}$$

Note that $E[V_x]$ equals the probability that V_x is 1. We compute an upper bound on this probability. From our earlier observation, it follows that if node x is visited and does not cause the algorithm to terminate, then q lies inside the hypercube of size $(1+2\sqrt{d}/\epsilon)s_x$, whose center coincides with the center of R_x. Since q is sampled from the uniform distribution in U, the volume of this hypercube, $(1+2\sqrt{d}/\epsilon)^d s_x^d$, is

an upper bound on the desired probability and hence on $E[V_x]$. Using this bound in (1) completes the proof. □

5. Logarithmic bound on expected query time of sliding-midpoint tree. By Lemma 4.1, the expected query time of priority search on a partition tree T is related to the quantities $\sum_{x\in\mathcal{I}_T} s_x^d$ and $\sum_{x\in\mathcal{L}_T} s_x^d$. The following lemma obtains upper bounds on these two quantities for the sliding-midpoint tree.

LEMMA 5.1. *Let S be a set of n data points in U. Let T be the partition tree for S built using the sliding-midpoint method. Then the following are true:* (i) $\sum_{x\in\mathcal{I}_T} s_x^d = O(\log n)$, *and* (ii) $\sum_{x\in\mathcal{L}_T} s_x^d = O(1)$. *The constant factors in the O-notation depend on d.*

Proof. Recall that the sliding-midpoint method partitions a cell into two subcells by a hyperplane orthogonal to its longest side and its middle. If there are data points in both subcells, it does nothing else. Otherwise, if there are no data points in one subcell, it slides the splitting plane until it just passes through a data point; the larger child becomes a leaf, and the smaller child becomes an internal node. It follows that the size of an internal node decreases by a factor of at least 2 as we descend d levels in the tree.

For $i \geq 1$, define *block* i to consist of levels $(i-1)d$ to $id-1$. Let $\mathcal{I}_i$ denote the set of internal nodes in block i. By the above observation, the size of an internal node in block i is at most $1/2^{i-1}$. Since the number of internal nodes in block i is at most $2^{(i-1)d}(2^d-1)$,

$$\sum_{x\in\mathcal{I}_i} s_x^d \leq \left(\frac{1}{2^{i-1}}\right)^d 2^{(i-1)d}(2^d-1) = 2^d - 1.$$

Let $\mathcal{I}'$ be the set of internal nodes in blocks numbered from 1 to $\lceil \log n/d \rceil$, and let $\mathcal{I}''$ be the remainder of the internal nodes. We will show that $\sum_{x\in\mathcal{I}'} s_x^d = O(\log n)$ and $\sum_{x\in\mathcal{I}''} s_x^d = O(1)$, which will prove (i).

Since $\mathcal{I}'$ consists of $\lceil \log n/d \rceil$ blocks,

$$\sum_{x\in\mathcal{I}'} s_x^d \leq \left\lceil \frac{\log n}{d} \right\rceil (2^d - 1) = O(\log n).$$

It remains to show that $\sum_{x\in\mathcal{I}''} s_x^d = O(1)$. Since each internal node has at least two data points, the number of internal nodes at any level is at most $\lceil n/2 \rceil \leq n$. Thus the number of internal nodes in a block is at most nd, which implies that

$$\sum_{x\in\mathcal{I}_i} s_x^d \leq nd \left(\frac{1}{2^{i-1}}\right)^d.$$

Therefore,

$$\sum_{x\in\mathcal{I}''} s_x^d \leq \sum_{i>\left\lceil \frac{\log n}{d} \right\rceil} \frac{nd}{2^{(i-1)d}} \leq nd \left(\frac{1/n}{1-1/2^d}\right) \leq 2d = O(1).$$

This completes the proof of (i).

Next we prove (ii). Recall that for any node x in the tree, R_x is the associated rectangle, defined by the splitting planes used in the construction of the tree. Now we associate a new rectangle R'_x with node x using the following simple procedure.

The root is associated with the unit hypercube U. Inductively, assume that R'_x is the rectangle associated with node x. If x is an internal node, then split any of the longest sides of R'_x at its middle, and associate the two resulting rectangles with the children of node x.

For each node x, let s'_x denote the longest side of the rectangle R'_x. Since the procedure splits the longest side of the rectangle each time, s'_x decreases by a factor of 2 as we descend d levels in the tree. Thus for any internal node x, $s_x \leq s'_x$, and for any leaf node x, $s_x \leq 2s'_x$, which implies that

$$\sum_{x \in \mathcal{L}_T} s_x^d \leq 2^d \sum_{x \in \mathcal{L}_T} (s'_x)^d. \tag{2}$$

Further, since the rectangles R'_x have an aspect ratio bounded by 2,

$$\sum_{x \in \mathcal{L}_T} (s'_x)^d \leq 2^{d-1} \sum_{x \in \mathcal{L}_T} v'_x, \tag{3}$$

where v'_x denotes the volume of rectangle R'_x. Also, $\sum_{x \in \mathcal{L}_T} v'_x = 1$, since the rectangles R'_x associated with the leaves form a subdivision of U. Combining this with (2) and (3), we get $\sum_{x \in \mathcal{L}_T} s_x^d = O(1)$. □

Using the upper bounds obtained in Lemma 5.1, it is easy to bound the expected query time.

THEOREM 5.2. *Let S be any set of n data points in $U = [0,1]^d$. Given any ϵ, assuming that the query point is sampled from the uniform distribution in U, the expected query time of priority search on the sliding-midpoint tree is $O((1/\epsilon)^d \log n)$. The constant factor in the O-notation depends on d.*

Proof. By Lemma 3.2, the query time is $O(I + Ld + L \log I)$, where I and L are the number of internal and leaf nodes visited, respectively. For fixed d, the expected query time is given by $O(E[I] + E[L \log I])$. Since $I \leq n$, the expected query time can be written as $O(E[I] + (\log n)E[L])$. By Lemmas 4.1 and 5.1, $E[I] = O((1/\epsilon)^d \log n)$ and $E[L] = O((1/\epsilon)^d)$. Thus the expected query time is $O((1/\epsilon)^d \log n)$. □

6. Optimality of sliding-midpoint tree. If the data points are uniformly distributed, then $\Omega(\log n)$ is a lower bound on the expected query time in the decision tree model. Thus it follows from Theorem 5.2 that the sliding-midpoint tree achieves optimal expected query time for uniformly distributed data points (ignoring constant factors depending on d and ϵ). The question naturally arises whether the sliding-midpoint tree achieves optimal performance for other data distributions as well. We prove that this is indeed the case under reasonable assumptions.

Let $\mathcal{T}_S$ denote the class of partition trees for S formed by splits using arbitrarily oriented hyperplanes (that is, *binary space partition trees*) such that each leaf cell contains at most one data point. We have the following result.

THEOREM 6.1. *Let d and ϵ be any fixed constants. There exists a constant c depending on d and ϵ such that the following is true. Let S be any set of n data points in $U = [0,1]^d$, and let T be any partition tree in $\mathcal{T}_S$. Assuming that the query point is sampled from the uniform distribution in U, the expected query time of priority search on the sliding-midpoint tree is no more than c times the expected query time of priority search on T.*

Let $\mathcal{Z}$ denote any set of regions inside U. (Note that the regions need not form a subdivision of U, nor are they necessarily disjoint.) We define the *entropy* of $\mathcal{Z}$ to be $\sum_{z \in \mathcal{Z}} v_z \log(1/v_z)$. The entropy of a subdivision of U is defined to be the entropy of the set of regions that form the subdivision.

We give a brief overview of the proof. Let S be a set of n data points in U. Let $\mathcal{D}$ denote any set of cells in U that satisfy the following properties for some constants c_a and c_n, depending on dimension.

A.1. *Difference of two rectangles:* A cell is the set-theoretic difference of two rectangles, one enclosed within the other. Note that the inner rectangle need not be present.

A.2. *Bounded aspect ratio:* The outer rectangle and inner rectangle (if present) have an aspect ratio (ratio of longest to shortest side) of at most c_a.

A.3. *Stickiness:* If the cell has an inner rectangle, then for each dimension, the separation between the corresponding faces of the inner and outer rectangles is either 0 or at least the length of the inner rectangle along that dimension.

A.4. *Existence of a close data point:* There is a data point whose distance from any point inside the outer rectangle of the cell is at most $c_n s$, where s is the size of the cell (that is, the length of the longest side of its outer rectangle).

A.5. *Disjointedness:* Given any two cells in $\mathcal{D}$, either the outer rectangles of the two cells are disjoint or the outer rectangle of one cell is contained within the inner rectangle of the other.

The basic idea of the proof is to show that the entropy of any such set of cells is a lower bound on the expected query time of priority search on any partition tree. For the upper bound, we will determine a set $\mathcal{D}$ of cells in U satisfying properties A.1–A.5 such that the expected query time of priority search on the sliding-midpoint tree is no more than the entropy of $\mathcal{D}$ (ignoring constant additive and multiplicative factors depending on d and ϵ).

6.1. Lower bound. The main result of this subsection is the following lemma.

LEMMA 6.2. *Let d and ϵ be any fixed constants. Let S be any set of n data points in U, and let T be any partition tree in $\mathcal{T}_S$. Let $\mathcal{D}$ be any set of cells in U satisfying properties* A.1–A.5. *Assuming that the query point is sampled from the uniform distribution in U, the expected query time of priority search on T is $\Omega(entropy(\mathcal{D}) + 1)$. The constant factor in the Ω-notation depends on d.*

Briefly, the proof works as follows. Let T be any partition tree in $\mathcal{T}_S$. Recall that $\mathcal{Z}_T$ denotes the subdivision of U induced by the leaf nodes of T. Since each internal node of T splits the associated region into two parts by a hyperplane, it follows that the cells in $\mathcal{Z}_T$ are convex polytopes. Further, each cell in $\mathcal{Z}_T$ either is empty or contains one data point. Let $\mathcal{Z}'_T$ denote the subdivision formed by splitting each nonempty cell in $\mathcal{Z}_T$ into two parts by passing a hyperplane through the data point inside it. The cells in $\mathcal{Z}'_T$ are convex polytopes in U and satisfy the following properties for constant c_v, depending on dimension (property B.1 is obvious; property B.2 is proved in Lemma 6.3).

B.1. *Empty interior:* A cell contains no data point in its interior.

B.2. *Proportionality of swept volume to radius:* Let z denote the cell, $\ominus$ denote the Minkowski difference operator, and B_r denote a ball of radius r. For any $r \geq 0$, the volume of $z - (z \ominus B_r)$ (that is, the set of points inside z within distance r of the boundary of z) is at most $c_v r$.

Let $\mathcal{D}$ be any set of cells in U satisfying properties A.1–A.5, and let $\mathcal{Z}$ be any subdivision of U into cells satisfying properties B.1–B.2. Lemmas 6.4, 6.5, and 6.6 form the cornerstone of the lower bound argument and show that the entropy of $\mathcal{D}$ is $O(entropy(\mathcal{Z})+1)$. Since $\mathcal{Z}'_T$ satisfies properties B.1–B.2, it follows that the entropy of $\mathcal{D}$ is $O(entropy(\mathcal{Z}'_T)+1)$. It is easy to show that $entropy(\mathcal{Z}'_T) \leq entropy(\mathcal{Z}_T)+1$. Thus $entropy(\mathcal{D}) = O(entropy(\mathcal{Z}_T)+1)$, as shown in Lemma 6.7. Finally, Lemma 6.8 proves

that the expected query time of priority search on T is $\Omega(\mathit{entropy}(\mathcal{Z}_T)+1)$. Together these imply that the expected query time of priority search on T is $\Omega(\mathit{entropy}(\mathcal{D})+1)$.

We start by proving that any convex polytope in U satisfies property B.2 (for constant $c_v = 2d$).

LEMMA 6.3. *Let z be a convex polytope contained within U. For any $r \geq 0$, the volume of $z - (z \ominus B_r)$ is at most $2dr$.*

Proof. Let $y = z - (z \ominus B_r)$. By definition, y is the set of points inside z at distance at most r from the boundary of z. For $d = 1$, z is a line segment, and it is obvious that the volume of y is at most $2r$.

Let $d > 1$. For each facet of z, construct a prism with this facet as base, directed inward into z, and with height r. It is easy to see that any point in y must lie in one of these prisms. Thus the volume of y is no more than the sum of the volume of all the prisms, which is clearly $a_z r$, where a_z is the surface area (perimeter, in two dimensions) of z. Since $z \subseteq U$, a_z is no more than the surface area of U, which is $2d$. (Here we have employed the following fact: if A and B are two closed, bounded, and convex subsets of R^d and $A \subseteq B$, then the surface area of A is no more than the surface area of B. See [13] for a proof.) Thus the volume of y is at most $2dr$. □

Now we are ready to present the key lemma for the lower bound argument. In view of the applications of the lemma to other problems (such as planar point location [1]), we prove a stronger version than is strictly needed here.

LEMMA 6.4. *Let d be any fixed constant. Let $\mathcal{Z}$ be any set of disjoint cells in U satisfying property* B.2 *(for constant c_v), and let $\mathcal{D}$ be any set of cells in U satisfying properties* A.1, A.2 *(for constant c_a),* A.3, *and* A.5. *Assume further that there exists a constant c_n such that, for any cell $u \in \mathcal{D}$ and $z \in \mathcal{Z}$, if $u_O \subseteq z$, then the distance to any point in u_O from the boundary of z is at most $c_n s_u$. Define a* fragment *to be a connected component in the intersection of a cell in $\mathcal{Z}$ with a cell in $\mathcal{D}$. Let $\mathcal{F}$ be the set of all fragments. Then*

$$\mathit{entropy}(\mathcal{F}) \leq d \cdot \mathit{entropy}(\mathcal{Z}) + O\left(\sum_{z \in \mathcal{Z}} v_z\right),$$

where the constant factor in the O-notation depends on d.

Proof. For any cell $z \in \mathcal{Z}$, let $\mathcal{F}_z \subseteq \mathcal{F}$ denote the set of fragments that are contained within z. We will show that $\mathit{entropy}(\mathcal{F}_z) \leq dv_z \log(1/v_z) + O(v_z)$. Since $\mathit{entropy}(\mathcal{F}) = \sum_{z \in \mathcal{Z}} \mathit{entropy}(\mathcal{F}_z)$, the lemma will then follow by summing over all $z \in \mathcal{Z}$.

Let z be any cell in $\mathcal{Z}$. We start by partitioning z into an infinite set of regions, denoted $z_i, i \geq 1$, as follows. Let $r_c = v_z/c_v$. Define z'_i to be the region consisting of points inside z at a distance of at least $r_c/2^i$ from the boundary of z. Clearly $z'_i \subseteq z'_{i+1}$ for $i \geq 1$. By property B.2 and the definition of r_c, it follows that the volume of z'_i is at least $v_z(1 - 1/2^i)$. The volume of z'_1 is thus at least $v_z/2$; we define z_1 to be any region of volume $v_z/2$ inside z'_1. Next observe that the volume of z'_2 is at least $3v_z/4$, and so the volume of $z'_2 - z_1$ is at least $v_z/4$; we define z_2 to be any region of volume $v_z/4$ inside z'_2 that is disjoint from z_1. Continuing in this way, we define $z_i, i \geq 1$, to be any region of volume $v_z/2^i$ inside z'_i that is disjoint from all of the regions $z_1, z_2, \ldots, z_{i-1}$. It is clear that the regions $z_i, i \geq 1$, form an infinite partition of z and satisfy the following two conditions: $v_{z_i} = v_z/2^i$ and $z_i \subseteq z'_i$.

Define a *subfragment* to be the intersection of a fragment in $\mathcal{F}_z$ with a region $z_i, i \geq 1$. Let $\mathcal{G}_z$ denote the set of all subfragments. Since each fragment $y \in \mathcal{F}_z$ is

partitioned into subfragments $y \cap z_i, i \geq 1$, it is easy to see that

$$entropy(\mathcal{F}_z) \leq entropy(\mathcal{G}_z). \tag{4}$$

Let $\mathcal{G}_{z_i}$ denote the set of subfragments that lie inside z_i. Clearly

$$entropy(\mathcal{G}_z) = \sum_{i \geq 1} entropy(\mathcal{G}_{z_i}). \tag{5}$$

Next we compute an upper bound on $entropy(\mathcal{G}_{z_i})$; the desired bound on $entropy(\mathcal{F}_z)$ will follow using (4) and (5). Obviously $entropy(\mathcal{G}_{z_i}) \leq entropy(\mathcal{G}_{z_i} \cup \{x\})$, where $x = z_i - \bigcup \mathcal{G}_{z_i}$ (that is, the region remaining in z_i after removing all the subfragments in $\mathcal{G}_{z_i}$). Let m_i be the number of subfragments in $\mathcal{G}_{z_i}$. It follows from basic calculus that $entropy(\mathcal{G}_{z_i} \cup \{x\})$ can be no more than the entropy of the set of regions formed by splitting z_i into $m_i + 1$ parts of equal volume. Thus

$$entropy(\mathcal{G}_{z_i}) \leq v_{z_i} \log \frac{m_i + 1}{v_{z_i}}. \tag{6}$$

We will show that $m_i \leq c \cdot 2^{id}/v_z^{d-1}$, where c is a constant that depends on dimension d. Assuming this fact for now and recalling that $v_{z_i} = v_z/2^i$, we get

$$entropy(\mathcal{G}_{z_i}) \leq \frac{v_z}{2^i} \log \frac{c2^{id}/v_z^{d-1} + 1}{v_z/2^i} \leq \frac{v_z}{2^i} \log \frac{(c+1)2^{id}/v_z^{d-1}}{v_z/2^i}.$$

Letting $c' = c + 1$ and simplifying, we get

$$entropy(\mathcal{G}_{z_i}) \leq \left(dv_z \log \frac{1}{v_z} + v_z \log c' \right) \frac{1}{2^i} + (d+1) v_z \frac{i}{2^i}.$$

Using (5), we obtain

$$entropy(\mathcal{G}_z) \leq dv_z \log \frac{1}{v_z} + v_z(\log c' + 2(d+1)) = dv_z \log \frac{1}{v_z} + O(v_z),$$

where the constant factor in the O-notation depends on d. By (4), $entropy(\mathcal{F}_z) \leq dv_z \log(1/v_z) + O(v_z)$, which is the desired claim.

It remains to show that $m_i \leq c \cdot 2^{id}/v_z^{d-1}$. Recall that m_i is the number of fragments that overlap z_i. Since $z_i \subseteq z_i'$, it follows that m_i is no more than the number of fragments that overlap z_i'. Let $\mathcal{F}_{z_i'}$ denote the set of fragments that overlap z_i'. For convenience, set $r_i = r_c/2^i$ and $c_n' = \max(c_n, \sqrt{d})$. Below we describe a set of disjoint hypercubes $\mathcal{H}$ and give a 1-1 mapping from $\mathcal{H}$ to $\mathcal{F}_{z_i'}$ such that at least a fraction $(1/2d)$ of the fragments in $\mathcal{F}_{z_i'}$ lie in the image of this mapping. (We will use H_y to denote the hypercube, if any, that maps to fragment y and say that H_y is *associated* with y.) Further, each hypercube in $\mathcal{H}$ is contained within z and has size $r_i/(c_n' c_a)$. A simple packing argument then implies that

$$|\mathcal{H}| \leq \frac{v_z}{(r_i/(c_n' c_a))^d}.$$

Substituting $r_i = r_c/2^i$ and $r_c = v_z/c_v$, we get

$$|\mathcal{H}| \leq \frac{(c_n' c_a c_v)^d 2^{id}}{v_z^{d-1}}.$$

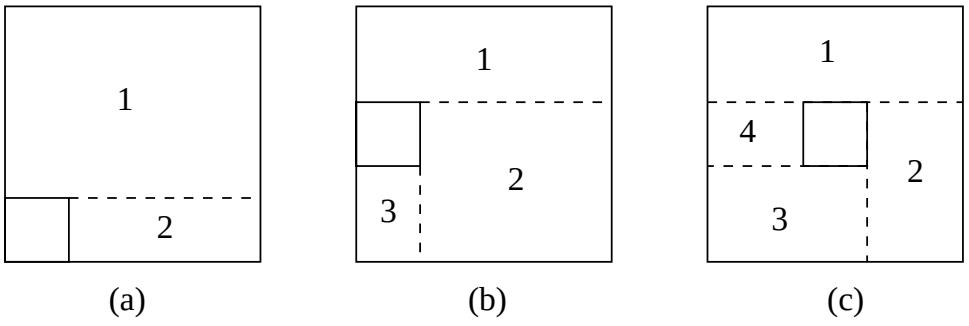


FIG. 3. *Partitioning a cell into rectangles.*

The number of fragments in $\mathcal{F}_{z'_i}$ is at most $2d$ times this quantity. We thus obtain the desired bound $m_i \leq c \cdot 2^{id}/v_z^{d-1}$, where $c = 2d(c'_n c_a c_v)^d$.

Let $u \in \mathcal{D}$ be any cell that contains a fragment of $\mathcal{F}_{z'_i}$. Clearly u overlaps z'_i. We claim that the size of u is at least r_i/c'_n. If u_O overlaps the boundary of z, then the diameter of u_O must exceed r_i, and so s_u is at least $r_i/\sqrt{d}$. Otherwise, u_O is contained within z, and, by the statement of the lemma, the distance to any point in u_O from the boundary of z is at most $c_n s_u$. Since u_O overlaps z'_i, it contains a point at a distance of at least r_i from the boundary of z; thus $c_n s_u \geq r_i$, which implies that $s_u \geq r_i/c_n$. Combining the two cases, we get $s_u \geq r_i/c'_n$, where $c'_n = \max(c_n, \sqrt{d})$. We will make use of this fact in our proof.

We now describe how to associate hypercubes with fragments in $\mathcal{F}_{z'_i}$; later we will show that the resulting set of hypercubes $\mathcal{H}$ satisfies all the properties mentioned earlier. For each fragment $y \in \mathcal{F}_{z'_i}$, choose a point p_y in $y \cap z'_i$. (We call such a point a *fragment representative point*.) Let $u \in \mathcal{D}$ be any cell that overlaps z'_i. We consider three cases: (i) u has no inner rectangle, (ii) u has an inner rectangle of size at least r_i/c'_n, and (iii) u has an inner rectangle of size less than r_i/c'_n.

If u has no inner rectangle, then, with each fragment $y \in \mathcal{F}_{z'_i}$ such that $y \subseteq u$, associate a hypercube H_y of size $r_i/(c'_n c_a)$ that overlaps p_y and is contained within u. Note there exists such a hypercube since, as shown above, $s_u \geq r_i/c'_n$ and, by property A.2 (bounded aspect ratio), the length of each side of u is at least $r_i/(c'_n c_a)$.

In the second case, u has an inner rectangle of size at least r_i/c'_n. Observe that u can be partitioned into t rectangles, where $1 \leq t \leq 2d$ is the number of sides of the inner rectangle that do not touch the corresponding side of the outer rectangle. (This can be done by successively passing hyperplanes that touch a side of the inner rectangle. For example, Figures 3(a), (b), and (c) show how a cell in two dimensions is partitioned into 2, 3, and 4 rectangles, respectively.) Note that by properties A.2 (bounded aspect ratio) and A.3 (stickiness), each side of these t rectangles is of length at least $r_i/(c'_n c_a)$. Of these t rectangles, let R denote the rectangle that has the maximum number of fragment representative points corresponding to the fragments in $\mathcal{F}_{z'_i}$. With each fragment $y \in \mathcal{F}_{z'_i}$ such that $p_y \in R$, associate a hypercube H_y of size $r_i/(c'_n c_a)$ that overlaps p_y and is contained within R. With the remaining fragments in u (that is, fragments whose corresponding representative point lies in $u - R$), we associate no hypercube.

In the third case, u has an inner rectangle of size less than r_i/c'_n. As in the second case, we partition u into t rectangles, $1 \leq t \leq 2d$, and determine the rectangle R among them that has the maximum number of fragment representative points

corresponding to the fragments in $\mathcal{F}_{z_i'}$. With each fragment $y \in \mathcal{F}_{z_i'}$ such that $p_y \in R$, associate a hypercube H_y of size $r_i/(c_n' c_a)$ that overlaps p_y and is contained within the outer rectangle of u. Such a hypercube exists because, as shown above, $s_u \geq r_i/c_n'$. With the remaining fragments in u, we associate no hypercube. (Note that, unlike the second case, the sides of R may have length less than $r_i/(c_n' c_a)$, and the hypercube H_y may overlap the inner rectangle of u.)

It remains to argue that the resulting set of hypercubes $\mathcal{H}$ has the desired properties. Let y denote a fragment in $\mathcal{F}_{z_i'}$. By construction, the size of H_y (if defined) is $r_i/(c_n' c_a)$. Second, since H_y overlaps p_y and p_y is at distance at least r_i from the boundary of z, it follows that H_y is contained within z. Third, for any cell u that overlaps z_i', it is clear that we associate a hypercube with at least a fraction $1/(2d)$ of the fragments of $\mathcal{F}_{z_i'}$ contained in u. It follows that a hypercube is associated with at least a fraction $1/(2d)$ of the fragments in $\mathcal{F}_{z_i'}$.

The only thing left to show is that the hypercubes are all disjoint. Let $y_1, y_2 \in \mathcal{F}_{z_i'}$ be any two distinct fragments that each have a hypercube associated with them. Let $u_1, u_2 \in \mathcal{D}$ be the cells containing y_1, y_2, respectively. There are four different cases: (i) $u_{1O} \cap u_{2O} = \emptyset$, (ii) $u_{2O} \subseteq u_{1I}$, (iii) $u_{1O} \subseteq u_{2I}$, and (iv) $u_1 = u_2$. (Note that, in case (iv), y_1 and y_2 are contained in the same cell.)

By our construction, $H_{y_1} \subseteq u_{1O}$ and $H_{y_2} \subseteq u_{2O}$. It follows that in case (i), H_{y_1} and H_{y_2} must be disjoint. In case (ii), recall that a cell containing a fragment of $\mathcal{F}_{z_i'}$ must have size at least r_i/c_n'. Thus u_2 has size at least r_i/c_n', and hence u_{1I} has size at least r_i/c_n'. By our construction, this implies that $H_{y_1} \subseteq u_1$. Since $H_{y_2} \subseteq u_{2O}$, it follows that H_{y_1} and H_{y_2} are disjoint. Case (iii) is similar to the second case. In case (iv), observe that the line segment joining p_{y_1} and p_{y_2} lies entirely within cell u_1. (This is obvious if u_1 has no inner rectangle; if u_1 has an inner rectangle, this follows from the fact that we partition u_1 into rectangles and associate hypercubes with fragment representative points lying in only one of these rectangles.) Since y_1 and y_2 are distinct fragments created by the intersection of z with u_1, the boundary of z must intersect this line segment. Since the distance of both p_{y_1} and p_{y_2} from the boundary of z is at least r_i, the distance between p_{y_1} and p_{y_2} must be at least $2r_i$. Since H_{y_1} and H_{y_2} overlap p_{y_1} and p_{y_2}, respectively, and each has size $r_i/(c_n' c_a)$, it is easy to verify that H_{y_1} and H_{y_2} are disjoint. This completes the proof of the lemma. □

LEMMA 6.5. *Let d be any fixed constant. Let $\mathcal{Z}$ and $\mathcal{D}$ be sets of cells satisfying all the conditions given in the statement of Lemma* 6.4. *Further, let $\mathcal{F}$ be as defined in the statement of Lemma* 6.4. *Assuming that $\mathcal{Z}$ is a subdivision of U,*

$$\mathit{entropy}(\mathcal{D}) \leq \mathit{entropy}(\mathcal{F}) \leq d \cdot \mathit{entropy}(\mathcal{Z}) + O(1),$$

where the constant factor in the O-notation depends on d.

Proof. Clearly, the set of fragments in $\mathcal{F}$ form a refinement of the set of cells in $\mathcal{D}$. Thus $\mathit{entropy}(\mathcal{D}) \leq \mathit{entropy}(\mathcal{F})$. Using Lemma 6.4 and the fact that $\sum_{z \in \mathcal{Z}} v_z = 1$, we get $\mathit{entropy}(\mathcal{F}) \leq d \cdot \mathit{entropy}(\mathcal{Z}) + O(1)$, which completes the proof. □

LEMMA 6.6. *Let d be any fixed constant. Let S be any set of n data points in U. Let $\mathcal{Z}$ be a subdivision of U whose cells satisfy properties* B.1–B.2, *and let $\mathcal{D}$ be a set of cells in U satisfying properties* A.1–A.5. *Then*

$$\mathit{entropy}(\mathcal{D}) = O(\mathit{entropy}(\mathcal{Z}) + 1),$$

where the constant factor in the O-notation depends on d.

Proof. Let u be a cell in $\mathcal{D}$ whose outer rectangle is contained within some cell $z \in \mathcal{Z}$. We claim that the distance to any point p in u_O from the boundary of z is at most $c_n s_u$. This follows from the facts that the distance between p and the nearest data point is at most $c_n s_u$ (property A.4) and z contains no data point in its interior (property B.1). Thus $\mathcal{D}$ and $\mathcal{Z}$ satisfy the conditions of Lemma 6.5, which implies the desired claim. □

LEMMA 6.7. *Let d be any fixed constant. Let S be any set of n data points in U, and let T be any partition tree in $\mathcal{T}_S$. Then entropy$(\mathcal{D}) = O($entropy$(\mathcal{Z}_T) + 1)$, where $\mathcal{D}$ is any set of cells in U satisfying properties* A.1–A.5. *The constant factor in the O-notation depends on d.*

Proof. Refine the subdivision $\mathcal{Z}_T$ by splitting each nonempty cell of $\mathcal{Z}_T$ into two parts by passing any hyperplane through the data point inside the cell. Let $\mathcal{Z}'_T$ be the new subdivision. It is easy to see that *entropy*$(\mathcal{Z}'_T)$ can be no more than the entropy of the set of regions formed by splitting each cell of $\mathcal{Z}_T$ into two parts of equal volume. Thus

$$\text{(7)}\quad \textit{entropy}(\mathcal{Z}'_T) \le \sum_{z \in \mathcal{Z}_T} 2\frac{v_z}{2} \log \frac{1}{v_z/2} = \sum_{z \in \mathcal{Z}_T} \left(v_z \log \frac{1}{v_z} + v_z \right) = \textit{entropy}(\mathcal{Z}_T) + 1,$$

where we have used the fact that $\sum_{z \in \mathcal{Z}_T} v_z = 1$.

Clearly, $\mathcal{Z}'_T$ is a subdivision of U into cells satisfying property B.1 (empty interior). Also, since the cells in $\mathcal{Z}'_T$ are convex polytopes contained within U, by Lemma 6.3, they satisfy property B.2 (proportionality of swept volume to radius). Thus $\mathcal{D}$ and $\mathcal{Z}'_T$ satisfy the conditions of Lemma 6.6, which implies that

$$\text{(8)}\qquad \textit{entropy}(\mathcal{D}) = O(\textit{entropy}(\mathcal{Z}'_T) + 1).$$

The lemma now follows from (7) and (8). □

LEMMA 6.8. *Let d and ϵ be any fixed constants. Let S be any set of n data points in U, and let T be any partition tree in $\mathcal{T}_S$. Assuming that the query point is sampled from the uniform distribution in U, the expected query time of priority search on T is $\Omega($entropy$(\mathcal{Z}_T) + 1)$.*

Proof. Recall that the algorithm starts by descending from the root of the tree T to the leaf that contains the query point. Thus the expected number of nodes visited by the algorithm to locate the leaf cell containing the query point is $\sum_{z \in \mathcal{Z}_T} v_z(\ell_z + 1)$. Note that this is the weighted external path length [20] of the tree, where the weight of a leaf is the volume of the associated cell. A fundamental information theoretic result due to Shannon [20, 25] implies that the weighted external path length of any binary tree with these weights is at least $\sum_{z \in \mathcal{Z}_T} v_z \log(1/v_z)$. Thus *entropy*$(\mathcal{Z}_T)$ is a lower bound on the expected number of nodes visited. Noting that the algorithm must visit at least one node, the lemma follows. □

Finally, Lemmas 6.7 and 6.8 together imply Lemma 6.2.

6.2. Upper bound. In this subsection, we compute an upper bound on the expected query time of priority search on the sliding-midpoint tree. We will prove that the expected query time is $O($*entropy*$(\mathcal{D}) + 1)$, where $\mathcal{D}$ is some set of cells in U satisfying properties A.1–A.5. In view of Lemma 6.2, this would imply the optimality of the sliding-midpoint tree in the sense of Theorem 6.1.

Our approach for obtaining $\mathcal{D}$ is as follows. In Lemma 6.9, we show that, for any sliding-midpoint tree T, there exists a closely related partition tree T' whose leaves satisfy properties A.1–A.5 and whose internal nodes satisfy properties A.1–A.4. In addition, T' possesses the following property, which is important for our analysis.

A.6. *Geometric decrease in volume:* The volume of the cells associated with the nodes decreases by at least a constant factor every $O(1)$ levels of descent in the tree. This property implies that if the volume of a leaf is v, then its level is $O(\log(1/v) + 1)$.

We define $\mathcal{D}$ to be the set of cells corresponding to the leaves of T'. In Lemmas 6.10, 6.11, 6.12, and 6.13, by exploiting properties of T and T', we give a simple analysis proving that the expected query time of priority search on T is $O(\mathit{entropy}(\mathcal{D}) + 1)$.

LEMMA 6.9. *Let S be a set of n data points in U. Let T be the sliding-midpoint tree for S. Then there exists a partition tree T' such that the following hold:*

(1) *The cells associated with the nodes (internal and leaf) of tree T' satisfy properties* A.1–A.4 *and* A.6. *Also, the set of cells associated with the leaves of T' satisfy property* A.5.

(2) *There exists a* 1-1 *mapping f from the nodes of T to the nodes of T' such that for any node x of T, $s_x/2 \le s_{f(x)} \le 8s_x$.*

Proof. We will construct the partition tree T' and the mapping f incrementally in $n-1$ steps. To this end, we label the internal nodes of T from 1 to $n-1$ such that the label assigned to any child is greater than the label assigned to its parent. Clearly, there exists such a labeling. (For example, label the internal nodes in a breadth-first manner.) Let T_i denote the subtree of T consisting of the root and the nodes whose parents have label $\le i$.

Let T'_i denote the tree constructed after i steps. In addition to this tree, we also maintain a 1-1 mapping f from the nodes of T_i to the nodes of T'_i. We will prove by induction that the following invariant holds at each step of the construction. (For convenience, we denote the cell associated with a node u by R_u if it is a rectangle and by C_u if it is the difference of two rectangles.)

(1) T'_i is a partition tree, and f is a 1-1 mapping from the nodes of T_i to the nodes of T'_i.

(2) Properties A.1–A.4 and A.6 hold for the cells associated with the nodes (internal and leaf) in T'_i.

(3) Property A.5 holds for the set of cells associated with the leaves of T'_i.

(4) Let x be any internal node in T that is present in T_i. Let $y = f(x)$. Then
   (a) the cell associated with node y is a rectangle (denoted R_y) and $R_x \subseteq R_y$,
   (b) each side of R_y has length $\ge s_x/2$ and $\le 4s_x$, and
   (c) if x is a leaf in T_i, then y is a leaf in T'_i.

(5) Let x be any leaf in T that is present in T_i. Let $y = f(x)$. Then $s_x/2 \le s_y \le 8s_x$.

It is clear that the lemma will then follow from the fact that the invariant holds for the final tree $T' = T'_{n-1}$.

Initially, T'_0 consists of just the root node, which is associated with U. Further, the root of T is mapped by f to the root of T'_0. It is easy to see that the invariant holds for $i = 0$. We now describe the ith step of the construction. Let x denote the internal node of T labelled i. Observe that our method of labeling implies that x is a leaf in T_{i-1}. Thus, by (4(c)) of the invariant, the corresponding node $y = f(x)$ must be a leaf in T'_{i-1}. Let x_1 and x_2 denote the two children of x. In the ith step, we *attach* a subtree consisting of a constant number of nodes to node y and map x_1 and x_2 by function f to two leaf nodes in this subtree. For the other nodes in T_i, the mapping f remains unchanged. (Note that they are all present in T_{i-1}.)

Now we give the details. Let the longest side of R_x that is split to form R_{x_1} and R_{x_2} be aligned along the kth coordinate axis, and let P denote the hyperplane

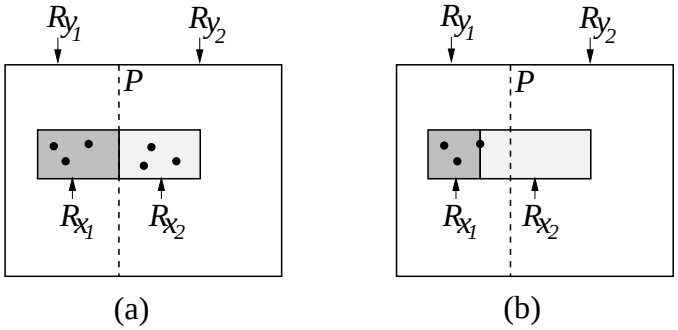


FIG. 4. *Proof of Lemma* 6.9.

orthogonal to it and passing through the center of R_x. We consider two cases.

Case 1. P is the splitting plane associated with node x. (See Figure 4(a).)

This implies that P splits R_x into R_{x_1} and R_{x_2} and there is a data point in each of these two rectangles. We modify T'_{i-1} in two substeps and show that the invariant holds after the second substep.

Substep 1. By (4(a)) of the invariant, the cell associated with y is a rectangle R_y, and $R_x \subseteq R_y$. We create two children y_1 and y_2 for node y. We then split R_y into two rectangles by hyperplane P and associate these rectangles with y_1 and y_2 such that R_{y_1} and R_{x_1} are both on the same side of P, and R_{y_2} and R_{x_2} are both on the other side of P. Henceforth, in our discussion of Case 1, we focus only on nodes x_1 and y_1 since nodes x_2 and y_2, respectively, play a symmetrical role.

We show that properties A.1–A.4 hold for R_{y_1}. First, observe that since R_{y_1} is a rectangle, it trivially satisfies properties A.1 (difference of two rectangles) and A.3 (stickiness). Second, since $R_{x_1} \subseteq R_{y_1}$ and there is a data point in R_{x_1}, it is clear that R_{y_1} satisfies property A.4 (existence of a close data point).

Third, we claim that each side of R_{y_1} has length $\geq s_x/2$ and $\leq 4s_x$. Note that this would imply that R_{y_1} satisfies property A.2 (bounded aspect ratio). From (4(b)) of the invariant, each side of R_y has length $\geq s_x/2$ and $\leq 4s_x$. Since R_{y_1} is formed from R_y by splitting side k (that is, the side aligned along the kth coordinate axis), the claim is obviously true for all sides of R_{y_1} other than side k. Further, the length of side k of R_{y_1} must be $\leq 4s_x$ since it must be smaller than the corresponding side of R_y. It remains to show that the length of side k of R_{y_1} is $\geq s_x/2$. Recall that $R_{x_1} \subseteq R_{y_1}$. Also, side k of R_{x_1} has length $s_x/2$ since P splits R_x at the middle of side k, which is one of the longest sides of R_x. Thus the length of side k of R_{y_1} is $\geq s_x/2$.

Next we claim that the volume of R_{y_1} is at most 7/8th the volume of R_y. As we will see, substep 2 creates either zero or two children for y_1; thus the volume of the nodes decreases by at least a factor of 8/7 after every two levels of descent in T'_i, which implies that T'_i satisfies property A.6 at the end of substep 2. To see the claim, observe that the ratio of the volume of R_{y_2} to the volume of R_y is the same as the ratio of the length of side k of R_{y_2} to the length of side k of R_y. Since the length of any side of R_{y_2} is $\geq s_x/2$, and the length of any side of R_y is $\leq 4s_x$, this ratio is at least 1/8. Thus the volume of R_{y_1} is at most 7/8th the volume of R_y.

Substep 2. Let $B_1 \subseteq R_{y_1}$ denote a rectangle formed by expanding each side of R_{x_1} that is smaller than $s_{x_1}/2$ to $s_{x_1}/2$. Note that it is possible to do the expansion such that B_1 lies inside R_{y_1}, since $R_{x_1} \subseteq R_{y_1}$ and, as shown in the discussion of substep 1, each side of R_{y_1} is of length $\geq s_x/2 \geq s_{x_1}/2$. Clearly, each side of B_1 has length

$\geq s_{x_1}/2$ and $\leq s_{x_1}$.

Next, if rectangle B_1 has a side that violates the stickiness property with respect to the corresponding side of R_{y_1}, expand it until it touches the side of R_{y_1}. Continue this process until all sides of the resulting rectangle (call it B_2) satisfy the stickiness property. It is easy to see that this can increase the length of a side of B_1 by a factor of at most 4 (since its length can increase by a factor of at most 2 in each of two expansions). Thus the length of each side of B_2 is $\geq s_{x_1}/2$ and $\leq 4s_{x_1}$.

If $B_2 = R_{y_1}$, then set $f(x_1) = y_1$. Otherwise, create two children y_{11} and y_{12} for node y_1, associate rectangle B_2 with y_{11} and $R_{y_1} - B_2$ with y_{12}, and set $f(x_1) = y_{11}$. Note that y_{12} will be a leaf in the final tree T'.

We now show that the invariant holds. We have already seen that T'_i satisfies property A.6, and properties A.1–A.4 hold for the node y_1 added in substep 1. We next show that properties A.1–A.4 hold for the children (if any) added to y_1.

If $f(x_1) = y_1$, then no children are added, and there is nothing to show. Otherwise, nodes y_{11} and y_{12} are made children of y_1. Recall that $R_{y_{11}} = B_2$ and $C_{y_{12}} = R_{y_1} - B_2$, and the length of each side of B_2 is $\geq s_{x_1}/2$ and $\leq 4s_{x_1}$. It follows that $R_{y_{11}}$ and $C_{y_{12}}$ satisfy properties A.1 (difference of two rectangles) and A.2 (bounded aspect ratio). Since B_2 is a rectangle, it satisfies property A.3 (stickiness). By construction of B_2, it is clear that $C_{y_{12}}$ also satisfies stickiness. Since B_2 has a data point inside it, it follows that B_2 and $C_{y_{12}}$ both satisfy property A.4 (existence of a close data point).

Next we show that (4(a)–(c)) of the invariant hold if x_1 is an internal node of T, and (5) of the invariant holds if x_1 is a leaf node. To prove (4(a)), note that by construction $R_{f(x_1)}$ is the rectangle B_2, and $R_{x_1} \subseteq B_2$ (since B_2 is formed by expanding R_{x_1}). Recall that each side of B_2 is of length $\geq s_{x_1}/2$ and $\leq 4s_{x_1}$, which implies (4(b)) and (5). Since $f(x_1)$ is a leaf in the tree T'_i, it follows that (4(c)) holds.

Finally, it is clear from our construction that T'_i is a partition tree whose leaves satisfy property A.5, and f is a 1-1 mapping from the nodes of T_i to the nodes of T'_i ((1) and (3) of the invariant).

Case 2. P is not the splitting plane associated with node x. (See Figure 4(b).)

This implies that all the data points in R_x are on the same side of P, and the splitting plane for x is obtained by sliding P along dimension k until it just passes through a data point. Without loss of generality, suppose that R_{x_1} is smaller than R_{x_2}, as shown in Figure 4(b). (The other case can be handled similarly.) Note that x_2 is a leaf containing exactly one data point.

As in Case 1, the cell associated with node y is a rectangle R_y and $R_x \subseteq R_y$. We modify T'_{i-1} in two substeps. In substep 1, we create two children y_1 and y_2 for node y. We then split R_y into two rectangles by hyperplane P; the rectangle that is on the same side of P as R_{x_1} is associated with y_1, and the other rectangle is associated with y_2. Next we set $f(x_2) = y_2$. Note that y_2 will be a leaf in the final tree T'.

The description of substep 2 is identical to Case 1, and so we omit it. Note that we need only to process nodes x_1 and y_1 in substep 2 (since x_2 is already mapped by f in substep 1).

We now show that the invariant holds. The argument is similar to that for Case 1, with two differences: (1) in showing that there is a data point close to R_{y_2} (property A.4), and (2) in showing that the size of R_{y_2} is $\geq s_{x_2}/2$ and $\leq 8s_{x_2}$ ((5) of the invariant).

Arguing as in Case 1, we can show that the length of each side of R_{y_2} is $\geq s_x/2$ and $\leq 4s_x$. Further, since R_x contains a data point (call it p), and $R_x \subseteq R_y$, the

distance between any point in R_y and p is no more than the diameter of R_y; by (4(b)) of the invariant, this is $\leq 4\sqrt{d}s_x$. Since $R_{y_2} \subseteq R_y$, this bound also applies to the distance between any point in R_{y_2} and p. It follows that R_{y_2} satisfies property A.4. Finally, to show the desired lower and upper bound on the size of R_{y_2}, note that R_{x_2} is the larger of the two children formed by applying the sliding-midpoint method to R_x. Thus $s_x \geq s_{x_2} \geq s_x/2$, which implies that the length of each side of R_{y_2} is $\geq s_{x_2}/2$ and $\leq 8s_{x_2}$. This completes the proof. □

The following lemma proved in [23] gives a bound on the number of leaf cells visited by priority search that holds irrespective of the data distribution and the location of the query point.

LEMMA 6.10 (see [23]). *The number of leaf cells of the sliding-midpoint tree visited by priority search in the worst case is* $O((1+1/\epsilon)^d)$. *The constant factor in the O-notation depends on d.*

Let T denote the sliding-midpoint tree, and let T' denote the tree corresponding to it, given in the statement of Lemma 6.9.

LEMMA 6.11. *The expected query time of priority search on the sliding-midpoint tree* T *is* $O(\sum_{x\in\mathcal{N}_{T'}} s_x^d)$. *The constant factor in the O-notation depends on d and* ϵ.

Proof. By Lemma 3.2, the query time is $O(I + Ld + L\log I)$, where I and L are the number of internal and leaf nodes visited, respectively. Thus, for fixed d, the expected query time is $O(E[I] + E[L\log I])$. By Lemma 6.10, L is bounded by a constant. Hence the expected query time is $O(E[I]+E[\log I]) = O(E[I])$. Lemma 4.1 implies that $E[I] = O(\sum_{x\in\mathcal{N}_T} s_x^d)$. By Lemma 6.9, there is a 1-1 function f that maps each node in T to a node in T', whose size is the same to within a constant factor. The lemma now follows. □

LEMMA 6.12. *Let* $\tilde{T}$ *be any partition tree in which the cells associated with the leaf and internal nodes of the tree satisfy properties* A.1–A.3 *and* A.6. *Let* $\mathcal{Z}_{\tilde{T}}$ *denote the subdivision of U induced by the leaf nodes of* $\tilde{T}$. *Then* $\sum_{x\in\mathcal{N}_{\tilde{T}}} s_x^d = O(entropy(\mathcal{Z}_{\tilde{T}}) + 1)$, *where the constant factor in the O-notation depends on d.*

Proof. Since the cells associated with the nodes of the tree satisfy property A.2 (bounded aspect ratio) and property A.3 (stickiness), it follows that $\sum_{x\in\mathcal{N}_{\tilde{T}}} s_x^d = O(\sum_{x\in\mathcal{N}_{\tilde{T}}} v_x)$. Since the volume of a node x is the sum of the volume of all the leaf nodes descended from it, we can write $\sum_{x\in\mathcal{N}_{\tilde{T}}} v_x = \sum_{x\in\mathcal{L}_{\tilde{T}}} v_x(\ell_x + 1)$, where ℓ_x denotes the level of leaf x. Further, by property A.6, $\ell_x = O(\log(1/v_x) + 1)$. Thus $\sum_{x\in\mathcal{N}_{\tilde{T}}} s_x^d = O(\sum_{x\in\mathcal{L}_{\tilde{T}}} v_x(\log(1/v_x) + 1)) = O(entropy(\mathcal{Z}_{\tilde{T}}) + 1)$. □

By Lemma 6.9, T' satisfies the conditions of Lemma 6.12, and thus $\sum_{x\in\mathcal{N}_{T'}} s_x^d = O(entropy(\mathcal{Z}_{T'}) + 1)$. Setting $\mathcal{D} = \mathcal{Z}_{T'}$ and applying Lemmas 6.9 and 6.11, we obtain the desired upper bound on the expected query time.

LEMMA 6.13. *The expected query time of priority search on the sliding-midpoint tree is* $O(entropy(\mathcal{D}) + 1)$, *where* $\mathcal{D}$ *is some set of cells in U satisfying properties* A.1–A.5. *The constant factor in the O-notation depends on d and* ϵ.

Finally, combining this lemma with the lower bound given in Lemma 6.2 establishes Theorem 6.1.

Remark. It is easy to see that our proof of Theorem 6.1 also implies the following: Let S be any set of n data points in U, and let A be any algorithm for finding the approximate nearest neighbor that can be modeled as an algebraic decision tree T using linear tests. Then the expected query time of priority search on the sliding-midpoint tree is no more than a constant (depending on d and ϵ) times the expected query time of A.

We mention only two key observations. First, the leaf nodes of T induce a subdivision $\mathcal{Z}_T$ of U into convex cells, such that each cell has at most one data point (since the same data point must be the approximate nearest neighbor, no matter where the query point lies in the cell). Second, the weighted path length of T, where the volume of a leaf is its weight, is a lower bound on the expected query time of A. Thus, as in section 6.1, we can prove that the expected query time of A is $\Omega(\mathit{entropy}(\mathcal{D}) + 1)$, where $\mathcal{D}$ is any set of cells in U satisfying properties A.1–A.5. The claim now follows in light of Lemma 6.13.

7. Experimental results. We ran experiments to compare the performance of the sliding-midpoint tree with the *standard kd-tree*, a partition tree devised by Friedman, Bentley, and Finkel [16] which is often used in nearest neighbor searching (both exact and approximate). The standard kd-tree recursively splits the data set into two sets of equal size by a hyperplane orthogonal to the dimension in which the points have maximum *spread* (difference of maximum and minimum coordinate). Friedman, Bentley, and Finkel showed that this tree can be used to answer nearest neighbor queries in $O(\log n)$ expected time, assuming that both data and query points are sampled from a distribution of bounded density.

We start by listing the point distributions used for generating the data sets [6, 24]. The clustered segments distribution was used to model data sets that exhibit clustering in low dimensional subspaces. The correlated Gaussian and correlated Laplacian point distributions were chosen to model data from speech processing applications. These two distributions were formed by grouping the output of autoregressive sources into vectors of length d. An autoregressive source uses the following recurrence to generate successive outputs:

$$X_n = \rho X_{n-1} + W_n,$$

where W_n is a sequence of zero mean independent, identically distributed random variables. The correlation coefficient ρ was taken as 0.9 for our experiments. Each point was generated by selecting its first coordinate from the corresponding uncorrelated distribution (either Gaussian or Laplacian), and then the remaining coordinates were generated by the equation above. See Farvardin and Modestino [14] for more information.

Uniform. Each coordinate was chosen uniformly from the interval $[0, 1]$.

Clustered segments. Eight axis-parallel line segments were sampled from a hypercube as follows. For each line segment a random coordinate axis x_k was selected, and a point p was sampled uniformly from the hypercube. The line segment is the intersection of the hypercube with the line parallel to x_k, passing through p. An equal number of points were generated uniformly along the length of each line segment and a Gaussian error with standard deviation of 0.001 was added.

Gaussian. Each coordinate was chosen from the Gaussian distribution with zero mean and unit variance.

Laplace. Each coordinate was chosen from the Laplacian distribution with zero mean and unit variance.

Correlated Gaussian. W_n was chosen so that the marginal density of X_n is normal with variance unity.

Correlated Laplacian. W_n was chosen so that the marginal density of X_n is Laplacian with variance unity.

We generated data points in dimension 16 from these distributions and in each case generated query points uniformly from a hypercube enclosing 90% of the data

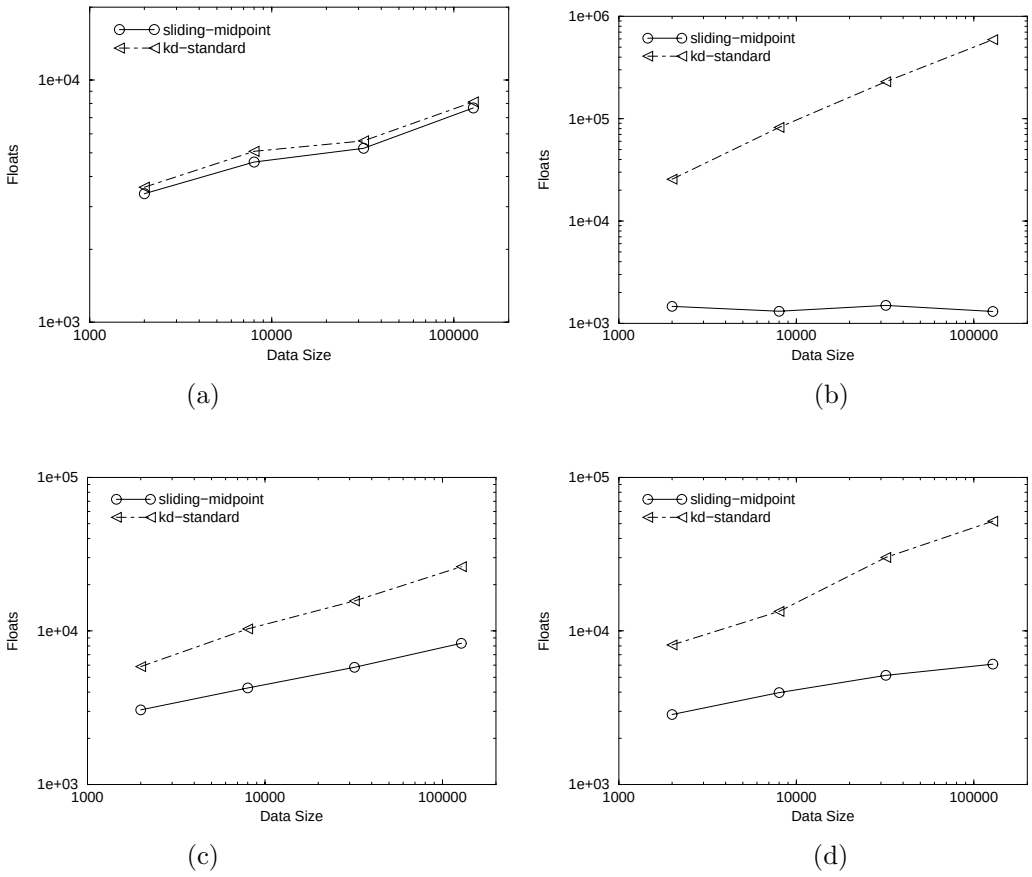


FIG. 5. *Average number of floating point operations for the* (a) *uniform,* (b) *clustered segments,* (c) *correlated Gaussian, and* (d) *correlated Laplacian distributions versus* n. *Here* $\epsilon = 2$.

points (to reduce the effect of outliers). Throughout, we used a bucket size (the maximum number of data points inside a leaf cell) of one for the partition trees.

For each experiment, we fixed ϵ and measured a number of statistics, averaging over 200 query points. The statistics included the average number of nodes visited, the average number of floating point operations (that is, any arithmetic operation involving point coordinates or distances), and the average CPU time. We present plots showing the number of floating point operations, which agrees well with the CPU times and provides a reasonable machine-independent measure of the query time.

The results for the uniform, clustered segments, correlated Gaussian, and correlated Laplacian distributions are shown in Figures 5 and 6. Figure 5 shows the average number of floating point operations as a function of n when ϵ is 2. Figure 6 shows the average number of floating point operations as a function of ϵ when n is $128,000$. We used a large value of ϵ in some of our experiments because we observed that the actual relative error committed by the algorithm is typically smaller by factors between 10 and 100.

We can make the following observations from the plots.

- The key observation is that for the clustered and the two correlated distri-

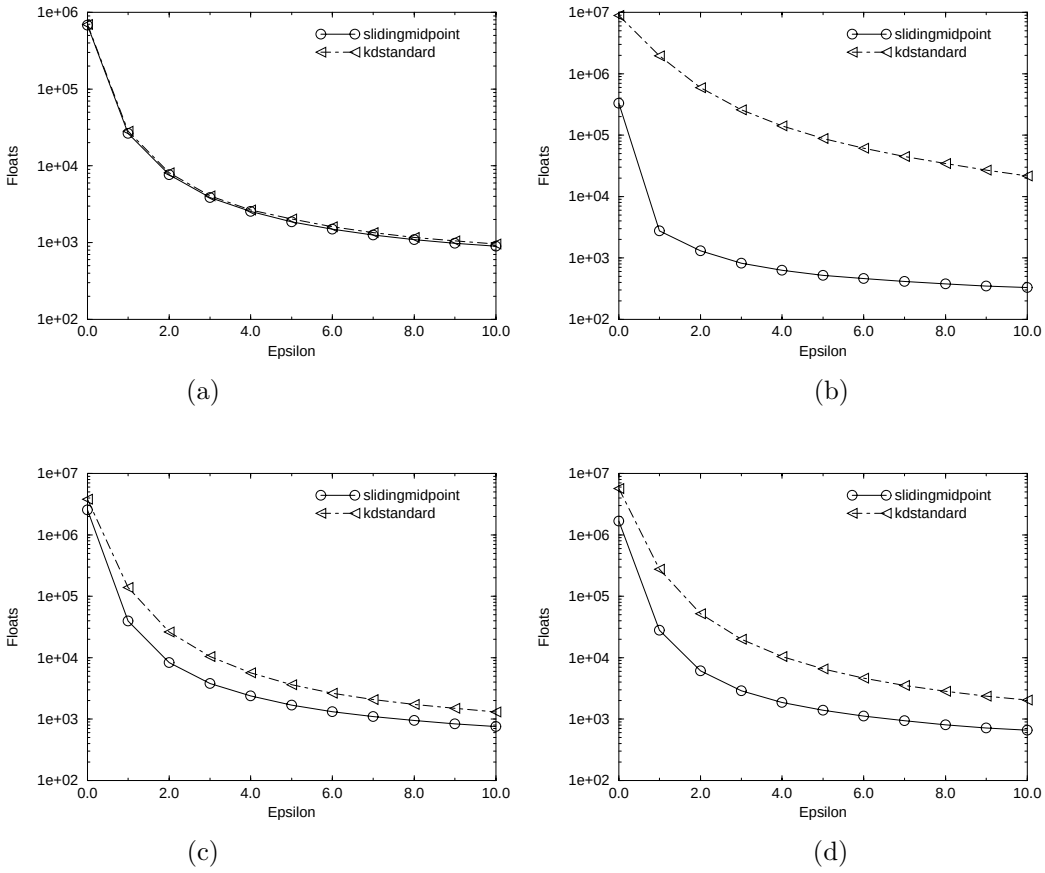


FIG. 6. *Average number of floating point operations for the* (a) *uniform,* (b) *clustered segments,* (c) *correlated Gaussian, and* (d) *correlated Laplacian distributions versus* ϵ. *Here* $n = 128,000$.

butions, the sliding-midpoint tree offers significant speed-up, sometimes by factors of over 10, compared to the standard kd-tree. Moreover, the speed-up increases with n.

- For the uniform distribution, both trees yield very similar query times.
- For the clustered segments distribution, the expected query time for the sliding-midpoint tree appears to be independent of the number of data points. (This is not hard to explain using Lemma 4.1.)
- As ϵ increases from 0 to 2, the query times of both the trees decrease significantly, usually by factors between 10 and 100.

Because of its design, the sliding-midpoint tree does a better job than the standard kd-tree of zooming toward the region where the data points are more densely clustered. This is the reason why it enjoys a considerable advantage for clustered data sets.

8. Conclusion. We have studied the approximate nearest neighbor problem from the perspective of expected-case performance. Our analysis assumes that the query points are sampled uniformly from a hypercube enclosing all the data points but makes no assumption on the distribution of data points. We have shown that the sliding-midpoint tree achieves linear space and logarithmic expected query time. We have also shown that this tree attains optimal expected query time (ignoring constant

factors) for any set of data points in a certain class of algorithms. The data structure is simple and easy to implement, and our empirical studies indicate that it performs well in practice.

There are several interesting open problems. The main limitation of our work is that it is restricted to the case of uniform query distribution. It would be interesting to develop an algorithm that achieves optimal expected query time for nonuniform query distributions. Another problem concerns strengthening the optimality claim for the sliding-midpoint tree. We proved that the sliding-midpoint tree achieves expected query time no more than a constant times that of any algorithm that can be modeled as an algebraic decision tree using linear tests. Does this optimality claim hold even if we allow polynomials of higher degree?

Acknowledgments. We would like to thank Siu-Wing Cheng, Mordecai Golin, Ramesh Hariharan, David Mount, Derick Wood, and the anonymous referees for many useful comments and suggestions.

REFERENCES

[1] S. Arya, S.-W. Cheng, D. M. Mount, and H. Ramesh, *Efficient expected-case algorithms for planar point location*, in Proceedings of the 7th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 1851, Springer-Verlag, Berlin, 2000, pp. 353–366.

[2] S. Arya and T. Malamatos, *Linear-size approximate Voronoi diagrams*, in Proceedings of the 13th ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2002, pp. 147–155.

[3] S. Arya, T. Malamatos, and D. M. Mount, *Space-efficient approximate Voronoi diagrams*, in Proceedings of the 34th ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 721–730.

[4] S. Arya and D. M. Mount, *Algorithms for fast vector quantization*, in Proceedings of DCC '93: Data Compression Conference, J. A. Storer and M. Cohn, eds., IEEE Computer Society, Los Alamitos, CA, 1993, pp. 381–390.

[5] S. Arya and D. M. Mount, *Approximate nearest neighbor queries in fixed dimensions*, in Proceedings of the 4th ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1993, pp. 271–280.

[6] S. Arya, D. M. Mount, N. Netanyahu, R. Silverman, and A. Y. Wu, *An optimal algorithm for approximate nearest neighbor searching in fixed dimensions*, J. ACM, 45 (1998), pp. 891–923.

[7] M. Bern, *Approximate closest-point queries in high dimensions*, Inform. Process. Lett., 45 (1993), pp. 95–99.

[8] T. M. Chan, *Approximate nearest neighbor queries revisited*, Discrete Comput. Geom., 20 (1998), pp. 359–373.

[9] K. L. Clarkson, *An algorithm for approximate closest-point queries*, in Proceedings of the 10th Annual ACM Symposium on Computational Geometry, ACM, New York, 1994, pp. 160–164.

[10] S. Deerwester, S. T. Dumals, G. W. Furnas, T. K. Landauer, and R. Harshman, *Indexing by latent semantic analysis*, J. Amer. Soc. Inform. Sci., 41 (1990), pp. 391–7407.

[11] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley, New York, 1973.

[12] C. A. Duncan, M. T. Goodrich, and S. G. Kobourov, *Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees*, J. Algorithms, 33 (2001), pp. 303–333.

[13] H. G. Eggleston, *Convexity*, Cambridge University Press, Cambridge, UK, 1958.

[14] N. Farvardin and J. W. Modestino, *Rate-distortion performance of DPCM schemes for autoregressive sources*, IEEE Trans. Inform. Theory, 31 (1985), pp. 402–418.

[15] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, *Query by image and video content: The QBIC system*, IEEE Comput., 28 (1995), pp. 23–32.

[16] J. H. Friedman, J. L. Bentley, and R. A. Finkel, *An algorithm for finding best matches in logarithmic expected time*, ACM Trans. Math. Software, 3 (1977), pp. 209–226.

[17] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic, Boston, MA, 1991.

[18] P. Indyk and R. Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 604–613.

[19] J. M. Kleinberg, *Two algorithms for nearest-neighbor search in high dimension*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 599–608.

[20] D. E. Knuth, *Sorting and Searching*, 2nd ed., The Art of Computer Programming 3, Addison–Wesley, Reading, MA, 1998.

[21] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, *Efficient search for approximate nearest neighbor in high dimensional spaces*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 614–623.

[22] S. Maneewongvatana and D. M. Mount, *Analysis of approximate nearest neighbor searching with clustered point sets*, in Data Structures, Near Neighbor Searches and Methodology: Fifth and Sixth DIMACS Implementation Challenges, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 59, M. H. Goldwasser, D. S. Johnson, and C. C. McGeoch, eds., AMS, Providence, RI, 2002, pp. 105–123.

[23] S. Maneewongvatana and D. M. Mount, *It's okay to be skinny, if your friends are fat*, in Proceedings of the 4th Annual CGC Workshop on Computational Geometry, Johns Hopkins University, Baltimore, MD, 1999, http://www.cs.jhu.edu/~cgc/abstracts99/mount.ps.

[24] D. M. Mount and S. Arya, *ANN: A library for approximate nearest neighbor searching*, in Proceedings of the 2nd Annual CGC Workshop on Computational Geometry, Duke University, Durham, NC, 1997, http://www.cs.duke.edu/CGC/workshop97.html.

[25] C. E. Shannon, *A mathematical theory of communication*, Bell System Tech. J., 27 (1948), pp. 379–423, 623–656.

SIAM J. COMPUT.
Vol. 32, No. 3, pp. 816–832

THE ONLINE MEDIAN PROBLEM*

RAMGOPAL R. METTU† AND C. GREG PLAXTON‡

Abstract. We introduce a natural variant of the (metric uncapacitated) k-median problem that we call the online median problem. Whereas the k-median problem involves optimizing the simultaneous placement of k facilities, the online median problem imposes the following additional constraints: the facilities are placed one at a time, a facility cannot be moved once it is placed, and the total number of facilities to be placed, k, is not known in advance. The objective of an online median algorithm is to minimize the competitive ratio, that is, the worst-case ratio of the cost of an online placement to that of an optimal offline placement. Our main result is a constant-competitive algorithm for the online median problem running in time that is linear in the input size. In addition, we present a related, though substantially simpler, constant-factor approximation algorithm for the (metric uncapacitated) facility location problem that runs in time linear in the input size. The latter algorithm is similar in spirit to the recent primal-dual-based facility location algorithm of Jain and Vazirani, but our approach is more elementary and yields an improved running time. While our primary focus is on problems which ask us to minimize the weighted average service distance to facilities, we also show that our results can be generalized to hold, to within constant factors, for more general objective functions. For example, we show that all of our approximation results hold, to within constant factors, for the k-means objective function.

Key words. approximation algorithms, k-median, facility location, discrete location theory, clustering, k-means

AMS subject classifications. 68W25, 68W40, 68Q25, 68Q17

PII. S0097539701383443

1. Introduction.

Suppose we wish to open a new chain of stores in a city with n neighborhoods and that we have a good estimate of the demand for our product in each neighborhood. In determining where to locate the stores, our high-level strategy is to minimize the *service cost* associated with our configuration of stores, which we define as the demand-weighted average distance from a customer to the nearest store. Our business plan is to start with one store and then to gradually add new stores as allowed by our profits. (Remark: We will never move a previously established store.) Thus our configuration of stores may change over time, and hence the ratio between the service cost of our configuration and that of an optimal same-size configuration may also change. The goal of the *online median problem* is to choose a site for each new store so that the maximum value of this ratio is minimized. An online median algorithm that guarantees a ratio of at most r is said to achieve a *competitive ratio* of r, or to be *r-competitive*.

The variant of this problem, in which the total number of stores to be built, k, is known in advance, corresponds to the classic *k-median problem*. The k-median problem is known to be $\mathcal{NP}$-hard and has been studied extensively over several decades (see, e.g., [25] for many pointers to the literature). Charikar et al. presented the

*Received by the editors January 9, 2001; accepted for publication (in revised form) January 10, 2003; published electronically April 23, 2003. This research was supported by NSF grant CCR-9821053.

http://www.siam.org/journals/sicomp/32-3/38344.html

†Department of Computer Science, Dartmouth College, Hanover, NH 03755 (ramgopal@cs.dartmouth.edu). This research was conducted while this author was at the University of Texas at Austin.

‡Department of Computer Science, University of Texas at Austin, Austin, TX 78712 (plaxton@cs.utexas.edu).

first polynomial-time constant-factor approximation algorithm for the k-median problem [5]; subsequently, improved time bounds and approximation factors have been obtained by Charikar and Guha [4], Jain and Vazirani [17], and Arya et al. [2].

Note that the online median problem can be viewed as the offline problem of determining a permutation of the n neighborhoods (specifying the order in which to build our stores) that minimizes the maximum ratio between the service cost of any prefix of the permutation and that of an optimal same-size configuration. We adopt this view throughout the remainder of the paper. Given the existence of constant-factor approximation algorithms for the k-median problem, it is natural to ask whether there is a constant-competitive algorithm for the online median problem. In other words, can we (efficiently) find a permutation of the n neighborhoods such that the service cost of any prefix of the permutation is at most a constant times that of an optimal same-size configuration? Note that, given an arbitrary problem instance, it is not clear a priori that such a permutation even exists.

In this paper, we affirm the existence of such a permutation and give a deterministic constant-competitive algorithm for the online median problem. Furthermore, the running time of our algorithm is $O(n^2 + \ell n)$ (where ℓ is the number of bits required to represent each distance), which is linear in the size of the input. While the main contribution of this paper is to identify and solve the online median problem, it worth noting that the k-median problem is a special case of the online median problem. Hence our linear-time online median algorithm is also the first deterministic constant-factor approximation algorithm for the k-median problem running in time that is linear in the size of the input. (The best previous running time of $O((n^2 \log n)(\ell + \log n))$ is given in [17].)

An obvious approach to the online median problem is to iteratively choose the point that minimizes the objective function. Greedy strategies of this kind are commonly applied in the design of online algorithms [3, 15]. It turns out, however, that for the online median problem, the simple strategy suggested above has an unbounded competitive ratio. We show that a modification of this strategy that we call *hierarchically greedy* can be used to obtain a constant-competitive algorithm for the online median problem that has a running time that is linear in the size of the input. We develop this strategy by first considering a simple greedy algorithm for facility location.

1.1. Problem definitions. Fix a set of points U, a distance function $d : U \times U \to \mathbb{R}$, and nonnegative functions $f, w : U \to \mathbb{R}$. We assume throughout that d is a metric, that is, d is nonnegative and symmetric and satisfies the triangle inequality, and $d(x, y) = 0$ iff $x = y$. For the online median problem, it will prove useful to consider a slightly more general class of distance functions in which the triangle inequality is relaxed to the following "λ-approximate" triangle inequality, where $\lambda \geq 1$: For any sequence of points $\langle x_0, \ldots, x_m \rangle$, $d(x_0, x_m) \leq \lambda \cdot \sum_{0 \leq i < m} d(x_i, x_{i+1})$. We refer to such a distance function as a λ-*approximate metric*. (Remark: The inequality associated with a λ-approximate metric is referred to as the "λ-polygonal inequality" in [9].) We let $n = |U|$, and we define a subset of U to be a *configuration* iff it is nonempty. For any point x and configuration X, we define $d(x, X)$ as $\min_{y \in X} d(x, y)$.

We consider three computational problems: k-median, online median, and facility location. For the k-median and online median problems, the *cost* of a configuration, which we denote as $cost(X)$, is defined to be $\sum_{x \in U} d(x, X) \cdot w(x)$. The input to the k-median problem is (U, d), w, and an integer k, $0 < k \leq n$. The output is a minimum-cost configuration of size k. The input to the online median problem is

(U, d) and w. The output is a total order on U. We define the competitive ratio of such an ordering as the maximum over all k, $0 < k \leq n$, of the ratio of the cost of the configuration given by the first k points in the ordering to that of an optimal k-median configuration. We define the *competitive ratio* of an online median algorithm as the supremum, over all possible choices of the input instance (U, d) and w, of the competitive ratio of the ordering produced by the algorithm.

For the facility location problem, the *cost* of a configuration, denoted $cost(X)$, is defined as the sum of $\sum_{x \in X} f(x)$ and $\sum_{x \in U} d(x, X) \cdot w(x)$. The input to the facility location problem is (U, d), f, and w. The output is a minimum-cost configuration.

1.2. Previous work. There has been much prior work on the facility location and k-median problems. In this paper, we focus on the metric versions of these problems; for recent work and pointers to the literature on the general (nonmetric) facility location and k-median problems, see [28]. The first constant-factor approximation algorithm for facility location is due to Shmoys, Tardos, and Aardal [26] and is based on rounding the (fractional) solution to a linear program (LP). Chudak [6] gives an LP-based $(1 + 2/e)$-approximation algorithm for facility location. This was the best constant factor known until the work of Charikar and Guha [4], which establishes a slightly lower approximation ratio of 1.728. Jain, Mahdian, and Saberi [16] give a simple greedy algorithm for the facility location that has an approximation ratio of 1.61. To our knowledge, the best approximation ratio for facility location is currently 1.52, due to Mahdian, Ye, and Zhang [23]. Guha and Khuller [12] provide the best lower bound known of 1.463 on the approximation ratio for the facility location problem.

The first constant-factor approximation for the k-median problem was recently given by Charikar et al. [5] and is also LP-based. That work follows a sequence of bicriteria results utilizing LP-based techniques [21, 22]. (These bicriteria results produce a configuration of size $O(k)$ with cost at most a constant factor times that of an optimal configuration of size k.) Jain and Vazirani [17] give the first nearly linear-time (in the input size) combinatorial algorithms for the facility location and k-median problems, achieving approximation ratios of 3 and 6, respectively. While the latter algorithms are combinatorial, the primal-dual approach used in their analysis is based on LP theory. (See [11] for an excellent introduction to the primal-dual method.) To our knowledge, the best approximation ratio for the k-median problem is $3 + \varepsilon$, due to Arya et al. [2]. Jain, Mahdian, and Saberi [16] provide the best lower bound known of $1 + 2/e$ on the approximation ratio for the k-median problem.

Strategies based on local search and greedy techniques for facility location and the k-median problem have previously been studied. The work of Korupolu, Plaxton, and Rajaraman shows that a simple local search heuristic proposed by Kuehn and Hamburger [20] yields both a constant-factor approximation for the facility location problem and a bicriteria approximation for the k-median problem [18]. To obtain their approximation result, Arya et al. [2] analyze a similar local search heuristic with a generalized local search step. Guha and Khuller [12] show that greedy improvement can be used as a postprocessing step to improve the approximation guarantee of certain facility location algorithms. Charikar and Guha [4] achieve an approximation ratio of 1.728 for facility location by combining a local search heuristic with the best LP-based algorithm known. Charikar and Guha also give a 4-approximation for the k-median problem by building on the techniques of Jain and Vazirani [17].

1.3. Contributions. Algorithms for problems in discrete location theory arise in many practical applications; see [7, 25], for example, for numerous pointers to the literature. Given that many of these problems are $\mathcal{NP}$-hard, it is desirable to

develop fast approximation algorithms. As mentioned above, it is not uncommon for approximation algorithms to be based on a greedy approach. In this paper, we show that greedy strategies yield a fast constant-factor approximation algorithm for the facility location problem and a fast constant-competitive algorithm for the online median problem.

We give an algorithm for the facility location problem that achieves an approximation ratio of 3 and runs in $O(n^2)$ time (i.e., time linear in the size of the input). The main idea of the algorithm is to compute and use the "value" of balls about every point in the metric space. In retrospect, the idea of value is implicit in the work of Jain and Vazirani [17]. We make this idea explicit and use the values of balls to make greedy choices. Additionally, our algorithm is faster than the Jain-Vazirani algorithm by a logarithmic factor.

While a simple greedy algorithm yields a constant-factor approximation bound for the facility location problem, it appears that a more sophisticated approach is needed to obtain a constant-factor approximation guarantee for the k-median problem, let alone a constant-competitiveness result for the online median problem. For example, in section 3, we show that perhaps the most natural greedy approach to the k-median (resp., online median) problem leads to an unbounded approximation (resp., competitive) ratio.

Our main result is a constant-competitive algorithm for the online median problem that runs in time linear in the size of the input. We achieve this result using a "hierarchically greedy" approach. The basic idea behind this approach is as follows: Rather than selecting the next point in the ordering based on a single greedy criterion, we greedily choose a region (the set of points lying within some ball) and then recursively select a point within that region. Thus the choice of point is influenced by a sequence of greedy criteria addressing successively finer levels of granularity.

Finally, we show that our analysis holds for more general classes of distance functions. We study two classes of "approximate" metrics for which the triangle inequality holds only to within a constant factor. We define and study *λ-approximate metrics* and *weakly λ-approximate metrics*. We show that our analysis holds to within constant factors given either of these two classes of distance functions. First, we show that λ-approximate distance functions facilitate an implementation of our online median algorithm running in time linear in the input size. We then show that weakly λ-approximate distance functions allow us to apply our techniques to objective functions other than the k-median objective. For example, we show that the approximation bounds for both of our algorithms hold to within constant factors for the well-known k-means objective function [8].

1.4. Outline. The rest of this paper is organized as follows. In section 2, we present our facility location algorithm and prove that it achieves an approximation ratio of 3. In section 3, we present our online median algorithm and prove that it is constant-competitive. Then, in section 4, we consider a weaker form of the triangle inequality in which we assume that the triangle inequality holds only to within a constant factor and show that our approximation bounds still hold (to within constant factors). Section 5 offers some concluding remarks.

2. Facility location. The following definitions are used throughout the present section as well as section 3.

(i) For any nonnegative integer m, let $[m]$ denote the set $\{i \mid 0 \leq i < m\}$.

(ii) A *ball* A is a pair (x, r), where the *center* x of A, denoted *center*(A), belongs to U, and the *radius* r of A, denoted *radius*(A), is a nonnegative real.

(iii) Given a ball $A = (x, r)$, we let *Points*(A) denote the set $\{y \in U \mid d(x, y) \leq r\}$. However, for the sake of brevity, we tend to write A instead of *Points*(A). For example, we write "$x \in A$" and "$A \cup B$" instead of "$x \in$ *Points*(A)" and "*Points*$(A) \cup$ *Points*(B)," respectively.

(iv) The *value* of a ball $A = (x, r)$, denoted *value*(A), is $\sum_{y \in A}(r - d(x, y)) \cdot w(y)$.

(v) For any ball $A = (x, r)$ and any nonnegative real c, we define cA as the ball (x, cr).

2.1. Algorithm. In the first step of the following algorithm, we assume that there is at least one point x such that $w(x) > 0$. (The problem is trivial otherwise.) The output of the algorithm is the configuration Z_n, which we also refer to as Z. (Remark: The indexing of the sets Z_i has been introduced solely to facilitate the analysis.)

1. For each point x, determine a ball $A_x = (x, r_x)$ such that *value*$(A_x) = f(x)$.
2. Determine a bijection $\varphi : [n] \to U$ such that $r_{\varphi(i-1)} \leq r_{\varphi(i)}$, $0 < i < n$.
3. Let $B_i = (x_i, r_i)$ denote the ball $A_{\varphi(i)}$, $0 \leq i < n$. Let $Z_0 = \emptyset$.
4. For $i = 0$ to $n - 1$: If $Z_i \cap 2B_i = \emptyset$, then let $Z_{i+1} = Z_i \cup \{x_i\}$; otherwise, let $Z_{i+1} = Z_i$.

We now sketch a simple $O(n^2)$-time implementation of the above algorithm. For each point x, the associated radius r_x can be computed in $O(n)$ time. (This is essentially a weighted selection problem.) Thus the first step requires $O(n^2)$ time. The second step involves sorting n values and can be accomplished in $O(n \log n)$ time. The running time for the third step is negligible. Each iteration of the fourth step can be easily implemented in $O(n)$ time; thus the time complexity of the fourth step is $O(n^2)$.

2.2. Approximation ratio. In this section, we establish the following theorem.

THEOREM 2.1. *For any configuration X, cost$(Z) \leq 3 \cdot$ cost(X).*

Proof. The proof is immediate from Lemmas 2.4 and 2.8 below. □

LEMMA 2.2. *For any point x_i, there exists a point x_j in Z such that $j \leq i$ and $d(x_i, x_j) \leq 2r_i$.*

Proof. If there is no such point x_j with $j < i$, then $Z_i \cap 2B_i$ is empty, and so x_i belongs to Z. □

LEMMA 2.3. *Let x_i and x_j be distinct points in Z. Then $d(x_i, x_j) > 2 \cdot \max\{r_i, r_j\}$.*

Proof. Assume without loss of generality that $j < i$. Thus $r_i \geq r_j$. Furthermore, $d(x_i, x_j) > 2r_i$ since x_j belongs to Z_i and $Z_i \cap 2B_i$ is empty. □

For any point x and any configuration X, let

$$\mathit{charge}(x, X) = d(x, X) + \sum_{x_i \in X} \max\{0, r_i - d(x_i, x)\}.$$

LEMMA 2.4. *For any configuration X, $\sum_{x \in U}$ charge$(x, X) \cdot w(x) =$ cost(X).*

Proof. Note that

$$\begin{aligned}\sum_{x \in U} \mathit{charge}(x, X) \cdot w(x) &= \sum_{x_i \in X} \sum_{x \in B_i} (r_i - d(x_i, x)) \cdot w(x) + \sum_{x \in U} d(x, X) \cdot w(x) \\ &= \sum_{x_i \in X} \mathit{value}(B_i) + \sum_{x \in U} d(x, X) \cdot w(x),\end{aligned}$$

which is equal to *cost*(X) since *value*$(B_i) = f(x_i)$. □

LEMMA 2.5. *Let x be a point, let X be a configuration, and let x_i belong to X. If $d(x, x_i) = d(x, X)$, then charge$(x, X) \geq \max\{r_i, d(x, x_i)\}$.*

Proof. If x does not belong to B_i, then $charge(x, X) \geq d(x, x_i) > r_i$. Otherwise, $charge(x, X) \geq d(x, x_i) + (r_i - d(x, x_i)) = r_i \geq d(x, x_i)$. □

LEMMA 2.6. *Let x be a point, and let x_i belong to Z. If x belongs to B_i, then $charge(x, Z) \leq r_i$.*

Proof. By Lemma 2.3, there is no point x_j in Z such that $i \neq j$ and x belongs to B_j. The claim now follows from the definition of $charge(x, Z)$, since $d(x, Z) \leq d(x, x_i)$. □

LEMMA 2.7. *Let x be a point, and let x_i belong to Z. If x does not belong to B_i, then $charge(x, Z) \leq d(x, x_i)$.*

Proof. The claim is immediate unless there is a point x_j in Z such that x belongs to B_j. If such a point x_j exists, then Lemmas 2.3 and 2.6 imply $d(x_i, x_j) > 2 \cdot \max\{r_i, r_j\}$ and $charge(x, Z) \leq r_j$, respectively. The claim now follows since $d(x, x_i) \geq d(x_i, x_j) - d(x, x_j) > 2r_j - r_j = r_j$. □

LEMMA 2.8. *For any point x and configuration X, $charge(x, Z) \leq 3 \cdot charge(x, X)$.*

Proof. Let x_i be some point in X such that $d(x, x_i) = d(x, X)$. By Lemma 2.2, there exists a point x_j in Z such that $j \leq i$ and $d(x_i, x_j) \leq 2r_i$.

If x belongs to B_j, then $charge(x, Z) \leq r_j$ by Lemma 2.6. The claim follows since $j \leq i$ implies $r_j \leq r_i$ and Lemma 2.5 implies $charge(x, X) \geq r_i$.

If x does not belong to B_j, then $charge(x, Z) \leq d(x, x_j)$ by Lemma 2.7. Thus $charge(x, Z) \leq d(x, x_i) + d(x_i, x_j) \leq d(x, x_i) + 2r_i$. The claim now follows by Lemma 2.5, since the ratio of $d(x, x_i) + 2r_i$ to $\max\{r_i, d(x, x_i)\}$ is at most 3. □

3. Online median placement. In the previous section, we found that a simple greedy algorithm yields interesting results for the facility location problem. The most obvious greedy algorithm for the online median problem is to select as the next point in the ordering the one that minimizes the objective function. Unfortunately, this algorithm gives an unbounded competitive (resp., approximation) ratio for the online median (resp., k-median) problem. To see this, consider an instance consisting of $n > 3$ points, one "red" and the rest "blue," such that the following conditions are satisfied: the red point has weight 0; each blue point has weight 1; the distance from the red point to any blue point is 1, and the distance between any pair of distinct blue points is 2. The aforementioned greedy algorithm chooses the red point first in the ordering, since that gives a cost of $n - 1$, while choosing any other point gives a cost of $2n - 4$. Consequently, the ratio for a configuration of size $n - 1$ is unbounded since the greedy cost is 1 and the optimal cost is 0. (This example also shows that no online median algorithm can achieve a competitive ratio below $2 - \frac{2}{n-1}$.)

We show that a more careful choice of the point, which we call hierarchically greedy, works well. Let Δ (resp., δ) denote the largest (resp., smallest) distance between two distinct points in the metric space. We define a certain ball about each point and select a ball A of maximum value. However, rather than simply choosing the center of ball A as the next point in the ordering, we apply the approach recursively to select a point within a region defined by A. At each successive level of recursion, we consider geometrically smaller balls about the remaining candidate points. Within $O(\log \frac{\Delta}{\delta})$ levels of recursion, we arrive at a ball containing a single point, and we return this point as the next one in the ordering. Note that whereas the greedy algorithm discussed in the previous paragraph makes a single greedy choice to select a point, the hierarchically greedy algorithm makes $O(\log \frac{\Delta}{\delta})$ greedy choices per point.

Throughout this section, let λ, α, β, and γ denote real numbers satisfying the following inequalities:

$$\lambda \geq 1, \tag{1}$$

$$\alpha > 1 + \lambda, \tag{2}$$

$$\beta \geq \frac{\lambda(\alpha - 1)}{\alpha - 1 - \lambda}, \tag{3}$$

$$\gamma \geq \left(\frac{\alpha^2 \beta + \alpha\beta}{\alpha - 1} + \alpha \right) \lambda. \tag{4}$$

The online median algorithm of section 3.1 below makes use of the following additional definitions.

(i) A *child* of a ball (x, r) is any ball $(y, \frac{r}{\alpha})$, where $d(x, y) \leq \beta r$.

(ii) For any point x and configuration X, let $\mathit{isolated}(x, X)$ denote the ball $(x, d(x, X)/\gamma)$. We let $\mathit{isolated}(x, \emptyset)$ denote the ball $(x, \max_{y \in U} d(x, y))$.

(iii) For any nonempty sequence ϱ, we let $\mathit{head}(\varrho)$ (resp., $\mathit{tail}(\varrho)$) denote the first (resp., last) element of ϱ.

3.1. Algorithm. Let $Z_0 = \emptyset$. For $i = 0$ to $n - 1$, execute the following steps:

1. Let σ_i denote the singleton sequence $\langle A \rangle$, where A is a maximum value ball in $\{\mathit{isolated}(x, Z_i) \mid x \in U \setminus Z_i\}$.

2. While the ball $\mathit{tail}(\sigma_i)$ has more than one child, append a maximum value child of $\mathit{tail}(\sigma_i)$ to σ_i.

3. Let $Z_{i+1} = Z_i \cup \{\mathit{center}(\mathit{tail}(\sigma_i))\}$.

The output of the online median algorithm is a collection of point sets Z_i such that $|Z_i| = i$, $0 \leq i \leq n$, and $Z_i \subseteq Z_{i+1}$, $0 \leq i < n$. Note that it is sufficient for an implementation of the algorithm to maintain the ball $\mathit{tail}(\sigma_i)$ as opposed to the entire sequence σ_i. The sequence σ_i has been introduced in order to facilitate the analysis.

We discuss two implementations of the online median algorithm in section 3.4. The first implementation has a running time that is slightly superlinear in the input size. The second implementation has a running time that is linear in the input size but assumes a (linear) preprocessing phase in which all distances are rounded down to the nearest integral power of λ. (Note that for the preprocessing phase to be well defined, we require $\lambda > 1$.) If the input distance function is a metric, it is straightforward to see that such rounding produces a λ-approximate metric.

3.2. Competitive ratio. Before proceeding with the analysis, we introduce a number of additional definitions.

(i) Let z_i denote the unique point in $Z_{i+1} \setminus Z_i$, $0 \leq i < n$.

(ii) For any configuration X and set of points Y, let $\mathit{cost}(X, Y) = \sum_{y \in Y} d(y, X) \cdot w(y)$.

(iii) For any configuration X, we partition U into $|X|$ sets $\{\mathit{cell}(x, X) \mid x \in X\}$ as follows: For each point y in U, we choose a point x in X such that $d(y, X) = d(y, x)$ and add y to $\mathit{cell}(x, X)$.

(iv) For any configuration X, point x in X, and set of points Y, we define $\mathit{in}(x, X, Y)$ as $\mathit{cell}(x, X) \cap \mathit{isolated}(x, Y)$ and $\mathit{out}(x, X, Y)$ as $\mathit{cell}(x, X) \setminus \mathit{in}(x, X, Y)$.

(v) For any configuration X and set of points Y, we let $\mathit{in}(X, Y)$ denote the set $\cup_{x \in X} \mathit{in}(x, X, Y)$ and $\mathit{out}(X, Y)$ denote $U \setminus \mathit{in}(X, Y)$.

Note that the $|X|$ sets $\mathit{cell}(x, X)$, $x \in X$, partition U by assigning each point in U to its closest point in X, breaking ties arbitrarily. The sets $\mathit{in}(x, X, Y)$ and $\mathit{out}(x, X, Y)$ partition the set $\mathit{cell}(x, X)$ into two disjoint sets. In our arguments, we will consider the sets $\mathit{in}(x, X, Z_{|X|})$ and $\mathit{out}(x, X, Z_{|X|})$ for $x \in X$, where X is an arbitrary configuration.

We note that the set $\mathit{out}(x, X, Z_{|X|})$ corresponds to the points in $\mathit{cell}(x, X)$ that are "outside" the ball $\mathit{isolated}(x, Z_{|X|})$. That is, if $\mathit{isolated}(x, Z_{|X|})$ has radius r,

then by the definition of $isolated(x, Z_{|X|})$, the points contained in $out(x, X, Z_{|X|})$ are exactly the points in $cell(x, X)$ that have distance greater than r to x but distance at most γr to some point in $Z_{|X|}$. Thus we can view the points in $out(x, X, Z_{|X|})$ as the points that are "close" to $Z_{|X|}$ and "far" from X. For any point y in $out(X, Z_{|X|})$, it is relatively straightforward (see Lemma 3.2) to show that $d(y, Z_{|X|})$ (i.e., the distance to the configuration $Z_{|X|}$ computed by our online median algorithm) is within a constant factor of $d(y, X)$.

We devote considerably more effort to showing that the cost incurred by $Z_{|X|}$ to serve the set $in(x, X, Z_{|X|})$ is within a constant factor of optimal. The set $in(x, X, Z_{|X|})$ corresponds to the points in $cell(x, X)$ that are contained in the ball $isolated(x, Z_{|X|})$. Suppose that $isolated(x, Z_{|X|})$ has radius r. By the definition of $isolated(x, Z_{|X|})$, the points contained in $in(x, X, Z_{|X|})$ are exactly the points in $cell(x, X)$ that are in the ball (x, r) but have distance strictly greater than γr to any point in $Z_{|X|}$. Thus the points in $in(x, X, Z_{|X|})$ are those points in $cell(x, X)$ that are "close" to X and "far" from $Z_{|X|}$. Accounting for the cost incurred by $Z_{|X|}$ for the points $in(X, Z_{|X|})$ will comprise the majority of the proofs in this subsection and the following subsection.

We now present our main result, Theorem 3.1. In order to minimize the competitive ratio of $2\lambda(\gamma + 1)$ implied by the theorem, we set λ to 1, set α to $2 + \sqrt{3}$, and set β and γ to the right-hand sides of (3) and (4), respectively. We thereby establish a competitive ratio of below 29.86 for the online median problem. In section 3.4, we describe an implementation of the online median algorithm for which the parameter λ is required to be strictly greater than 1. The degradation in the competitive ratio that results by setting λ greater than 1 can be made arbitrarily small by choosing λ sufficiently close to 1.

THEOREM 3.1. *For any configuration X, $cost(Z_{|X|}) \leq 2\lambda(\gamma + 1) \cdot cost(X)$.*

Proof. Let $Y = in(X, Z_{|X|})$, and let $Y' = out(X, Z_{|X|}) = U \setminus Y$. Note that $cost(X) = cost(X, Y) + cost(X, Y')$ and $cost(Z_{|X|}) = cost(Z_{|X|}, Y) + cost(Z_{|X|}, Y')$. Thus the theorem follows immediately from Lemmas 3.3, 3.5, and 3.6 below. ∎

LEMMA 3.2. *For any configuration X, and points x in X and y in $out(x, X, Z_{|X|})$, $d(y, Z_{|X|}) \leq \lambda(\gamma + 1) \cdot d(y, X)$.*

Proof. Let $isolated(x, Z_{|X|}) = (x, r)$. Note that $d(x, y) > r$. Also, by the definition of $isolated(x, Z_{|X|})$, there is a point z in $Z_{|X|}$ such that $d(x, z) = \gamma r$. Hence $d(y, z) \leq \lambda[d(x, y) + d(x, z)] = \lambda[d(x, y) + \gamma r] < \lambda[d(x, y) + \gamma \cdot d(x, y)] = \lambda(\gamma + 1) \cdot d(x, y) = \lambda(\gamma+1) \cdot d(y, X)$, where the last step follows since y is in $cell(x, X)$. The claim follows since $d(y, z) \geq d(y, Z_{|X|})$. ∎

LEMMA 3.3. *For any configuration X, $cost(Z_{|X|}, out(X, Z_{|X|}))$ is at most $\lambda(\gamma + 1) \cdot cost(X, out(X, Z_{|X|}))$.*

Proof. Summing the inequality of Lemma 3.2 over all y in $out(x, X, Z_{|X|})$, we obtain

$$cost(Z_{|X|}, out(x, X, Z_{|X|})) \leq \lambda(\gamma + 1) \cdot cost(X, out(x, X, Z_{|X|})).$$

The claim now follows by summing the above inequality over all x in X. ∎

LEMMA 3.4. *For any configuration X and point x in X, $cost(Z_{|X|}, in(x, X, Z_{|X|}))$ is at most $\lambda(\gamma + 1)[cost(X, in(x, X, Z_{|X|})) + value(isolated(x, Z_{|X|}))]$.*

Proof. Assume that $isolated(x, Z_{|X|}) = (x, r)$. Note that $d(x, y) = \gamma r$ for some y in $Z_{|X|}$. Thus, for any z in $isolated(x, Z_{|X|})$, $d(y, z) \leq \lambda[d(y, x) + d(x, z)] \leq \lambda(\gamma + 1)r$, where the last step follows from our bound on $d(x, y)$ and the definition of $isolated(x, Z_{|X|})$. It follows that $cost(Z_{|X|}, in(x, X, Z_{|X|}))$ is at most $\lambda(\gamma + 1)$

times

$$\sum_{z \in in(x,X,Z_{|X|})} r \cdot w(z) \leq \sum_{z \in in(x,X,Z_{|X|})} d(x,z) \cdot w(z) + \sum_{z \in isolated(x,Z_{|X|})} (r - d(x,z)) \cdot w(z)$$
$$= cost(X, in(x, X, Z_{|X|})) + value(isolated(x, Z_{|X|})). \qquad \square$$

LEMMA 3.5. *For any configuration X and point x in X, $cost(Z_{|X|}, in(X, Z_{|X|}))$ is at most $\lambda(\gamma+1)[cost(X, in(X, Z_{|X|})) + \sum_{x \in X} value(isolated(x, Z_{|X|}))]$.*

Proof. The claim follows by summing the inequality of Lemma 3.4 over all x in X. $\square$

Our main technical lemma is stated below. The proof is given in the next subsection.

LEMMA 3.6. *For any configuration X,*

$$\sum_{x \in X} value(isolated(x, Z_{|X|})) \leq cost(X).$$

3.3. Proof of Lemma 3.6. In this section, we establish our main technical lemma, Lemma 3.6. Informally, Lemma 3.6 yields an upper bound on the value of certain balls containing points "far" from $Z_{|X|}$, where X is an arbitrary configuration. The upper bound we obtain states that the value associated with these points is at most $cost(X)$. Thus, in combination with Lemmas 3.3 and 3.5, we can conclude that $cost(Z_{|X|})$ is $O(cost(X))$. To prove Lemma 3.6, we argue that for each ball $isolated(x, Z_{|X|})$, it is possible to identify a ball with commensurately high value that does not contain a point from X. More precisely, we construct a matching between the points in $Z_{|X|}$ and X and show that for each point x in $X \setminus Z_{|X|}$ we can identify a ball A_x appearing in some sequence $\sigma_i < |X|$ such that $value(A_x) \geq isolated(x, Z_{|X|})$, $cost(X, A_x) \geq value(A_x)$, and all such balls A_x are disjoint. Intuitively, we will identify these balls by making use of the greedy manner in which our online median algorithm constructs the sequences of balls σ_i, $0 \leq i < |X|$.

LEMMA 3.7. *Let $A = (x, r)$ belong to σ_i. Then $d(x, Z_i) \geq \gamma r$.*

Proof. Let z be a point in Z_i such that $d(x,z) = d(x, Z_i)$. If $A = head(\sigma_i)$, then $A = isolated(x, Z_i)$, and the result is immediate. Otherwise, let $B = (y, s)$ denote the predecessor of A in σ_i, and assume inductively that $d(y, Z_i) \geq \gamma s$. Note that $d(x,y) \leq \beta s$ and $s = \alpha r$. Thus $d(x, Z_i) = d(x,z) \geq d(y,z)/\lambda - d(x,y) \geq (\gamma/\lambda - \beta)\alpha r \geq \gamma r$, where the last step follows from (4). $\square$

LEMMA 3.8. *Let $A = (x, r)$ belong to σ_i, and let $B = (y, s)$ belong to σ_j. If $i < j$ and $d(x,y) \leq r + s$, then the following claims hold:* (i) *$radius(head(\sigma_j)) \leq \frac{r}{\alpha}$;* (ii) *$A \neq tail(\sigma_i)$;* (iii) *the successor of A in σ_i (call it C) satisfies $value(C) \geq value(head(\sigma_j))$.*

Proof. Let $head(\sigma_j) = (y', s')$. For part (i), we begin by deriving upper and lower bounds on $d(y', z_i)$. For a lower bound on $d(y', z_i)$, note that $d(y', z_i) \geq d(y', Z_j)$ (since $i < j$) and $d(y', Z_j) \geq \gamma s'$ by Lemma 3.7. To derive an upper bound on $d(y', z_i)$, we first let P denote the prefix of sequence σ_j ending with ball B, and we let S denote the suffix of sequence σ_i beginning with ball A. We then apply the λ-approximate triangle inequality to the sequence of points $\langle y', \ldots, y, x, \ldots, z_i \rangle$, where the prefix $\langle y', \ldots, y \rangle$ corresponds to the centers of the balls in P and the suffix $\langle x, \ldots, z_i \rangle$ corresponds to the centers of the balls in S. By repeated application of the definition of a child, and using the given upper bound on $d(x,y)$, we obtain

$$d(y', z_i) \leq \lambda \left[\beta \left(s' + \frac{s'}{\alpha} + \cdots + \alpha s \right) + s + r + \beta \left(r + \frac{r}{\alpha} + \cdots \right) \right]$$

$$\leq \left[\frac{\alpha\beta}{\alpha-1}\cdot(r+s')+r\right]\lambda.$$

Combining the bounds on $d(y', z_i)$ and applying (4), we obtain

$$\left(\frac{\alpha^2\beta+\alpha\beta}{\alpha-1}+\alpha\right)\lambda s' \leq \left[\frac{\alpha\beta}{\alpha-1}\cdot(r+s')+r\right]\lambda.$$

Multiplying through by $(\alpha-1)/\lambda$ and rearranging, we get $r \geq \frac{\alpha^2\beta+\alpha^2-\alpha}{\alpha\beta+\alpha-1}\cdot s' = \alpha s'$, establishing the claim.

For part (ii), note that $d(x,y) \leq r + \frac{r}{\alpha} < \beta r$ by part (i) and (3). Thus A has at least two children; the claim follows.

For part (iii), we obtain an upper bound on $d(x, y')$ by applying the λ-approximate triangle inequality to the sequence of points $\langle y', \ldots, y, x\rangle$, where the prefix $\langle y', \ldots, y\rangle$ corresponds to the centers of the balls in P (as defined in part (i) above). By repeated application of the definition of a child and by the given upper bound on $d(x,y)$, we observe that

$$d(x,y') \leq \lambda\left[r+s+\left(\alpha s+\alpha^2 s+\cdots+s'\right)\beta\right].$$

Then, by using (2) and (3) and part (i), we observe that

$$\begin{aligned}\lambda\left[r+s+\left(\alpha s+\alpha^2 s+\cdots+s'\right)\beta\right] &\leq \lambda r+\frac{\alpha\beta\lambda}{\alpha-1}\cdot s'\\ &\leq \lambda r+\frac{\alpha\beta\lambda}{\alpha-1}\cdot\frac{r}{\alpha}\\ &\leq \left(\frac{\beta}{\alpha-1}+1\right)\lambda r.\end{aligned}$$

Observe that $(\frac{\beta}{\alpha-1}+1)\lambda r$ is at most βr by (3). It then follows that $head(\sigma_j)$ is contained in a child of A. Thus $value(C) \geq value(head(\sigma_j))$. □

For ease of notation, throughout the remainder of this section, we fix a configuration X, and let k denote $|X|$. We now describe a *pruning procedure* that we use for the purpose of analyzing our online median algorithm. The pruning procedure takes as input the k sequences σ_i, $0 \leq i < k$, and produces as output k sequences τ_i, $0 \leq i < k$. The sequence τ_i is initialized to σ_i, $0 \leq i < k$. The (nondeterministic) pruning procedure then performs a number of iterations. In a general iteration, the pruning procedure checks whether there exist two balls $A = (x, r)$ and $B = (y, s)$ in distinct sequences τ_i and τ_j, respectively, such that $i < j$ and $d(x,y) \leq r + s$. If not, the pruning procedure terminates. If so, the sequence τ_i is redefined as the proper suffix of (the current) τ_i beginning at the successor of A. Note that part (ii) of Lemma 3.8 ensures that the pruning procedure is well defined. Furthermore, the procedure is guaranteed to terminate since each iteration reduces the length of some sequence τ_i.

LEMMA 3.9. *Let $A = (x, r)$ belong to τ_i, and let $B = (y, s)$ belong to τ_j. If $i < j$, then $d(x,y) > r + s$.*

Proof. The proof is immediate from the definition of the pruning procedure. □

LEMMA 3.10. *Each sequence τ_i is nonempty.*

Proof. The proof is immediate from part (ii) of Lemma 3.8 and the definition of the pruning procedure. □

LEMMA 3.11. *Let x be a point, and assume that $0 \leq i < j \leq n$. Then*

$$value(isolated(x, Z_i)) \geq value(isolated(x, Z_j)).$$

Proof. Since $Z_i \subseteq Z_j$, $radius(isolated(x, Z_i)) \geq radius(isolated(x, Z_j))$. The claim follows. ∎

LEMMA 3.12. *Let x be a point, and assume that $0 \leq i < k$. Then*

$$value(head(\sigma_i)) \geq value(isolated(x, Z_k)).$$

Proof. If x belongs to Z_i, then $radius(isolated(x, Z_i)) = 0$. It follows that $value(isolated(x, Z_i)) = 0$, and there is nothing to prove. Otherwise, $value(head(\sigma_i)) \geq value(isolated(x, Z_i))$ by the definition of the online median algorithm, and the claim follows by Lemma 3.11. ∎

LEMMA 3.13. *Let x be a point, and assume that $0 \leq i < k$. Then*

$$value(head(\tau_i)) \geq value(isolated(x, Z_k)).$$

Proof. We prove that the claim holds before and after each iteration of the pruning procedure. Initially, $\tau_i = \sigma_i$, and the claim holds by Lemma 3.12. If the claim holds before an iteration of the pruning procedure, then it holds after the iteration by part (iii) of Lemma 3.8. ∎

A ball $A = (x, r)$ is defined to be *covered* iff $d(x, X) < r$. A ball is *uncovered* iff it is not covered.

LEMMA 3.14. *For any uncovered ball $A = (x, r)$, $cost(X, A) \geq value(A)$.*

Proof. Note that $cost(X, A) \geq \sum_{y \in A} d(y, X) \cdot w(y) \geq \sum_{y \in A} (r - d(y, x)) \cdot w(y) = value(A)$. ∎

Let I denote the set of all indices i in $[k]$ such that some ball in τ_i is covered. We now construct a matching between the sets $[k]$ and X as follows. First, for each i in I, we match i with a point x in X that belongs to the last covered ball in the sequence τ_i. (Note that such a point x is guaranteed to exist by the definition of I. Furthermore, Lemma 3.9 ensures that we do not match the same point with more than one index.) Second, for each i in $[k] \setminus I$ in turn, we match i with an arbitrary unmatched point x in X.

We now construct a function φ mapping each point x in X to an uncovered ball. For each x in X that is matched with an index i in $[k] \setminus I$, we set $\varphi(x)$ to $head(\tau_i)$. For each x in X that is matched with an index i in I, we set $\varphi(x)$ to the successor of the last covered ball in τ_i unless $tail(\tau_i)$ is covered, in which case we set $\varphi(x)$ to the ball $(x, 0)$.

LEMMA 3.15. *For any pair of distinct points x and y in X, $\varphi(x) \cap \varphi(y) = \emptyset$.*

Proof. The proof is immediate from Lemma 3.9 and the fact that the ball $(x, 0)$ is contained in $tail(\tau_i)$. ∎

LEMMA 3.16. *For any point x in X, $value(\varphi(x)) \geq value(isolated(x, Z_k))$.*

Proof. If x is matched with an index i in $[k] \setminus I$, the claim follows by Lemma 3.13. If x is matched with an index i in I, we consider two cases. If $tail(\tau_i)$ is covered, then $x = z_i$ since $tail(\tau_i)$ has exactly one child. The claim follows since $\varphi(x) = isolated(x, Z_k) = (x, 0)$. If $tail(\tau_i)$ is uncovered, then the predecessor of $\varphi(x)$ in τ_i (call it $A = (y, r)$) exists and contains x. It follows that $value(\varphi(x)) \geq value(B)$, where $B = (x, r/\alpha)$ is the child of A centered at x. Let $C = (x, s)$ denote the ball $isolated(x, Z_k)$. Below we complete the proof of the claim by showing that $r/\alpha \geq s$, which implies that $B \supseteq C$ and hence $value(B) \geq value(C)$.

It remains to prove that $r/\alpha \geq s$ in the final case considered above. We prove the claim by deriving upper and lower bounds on $d(x, z_i)$. Let S be the suffix of the sequence τ_i beginning with the ball A. For the upper bound, we apply the triangle inequality to the sequence of points $\langle x, y, \ldots, z_i \rangle$, where the suffix $\langle y, \ldots, z_i \rangle$ consists of the centers of the balls in S. We then obtain that

$$\begin{aligned} d(x, z_i) &\leq \lambda \left(r + \beta \left(r + \frac{r}{\alpha} + \cdots \right) \right) \\ &\leq \left(1 + \frac{\alpha\beta}{\alpha - 1} \right) \lambda r, \end{aligned}$$

which is less than $\gamma r/\alpha$ by (4). The desired inequality follows since $d(x, z_i) \geq \gamma s$ by the definition of C. □

Lemmas 3.14, 3.15, and 3.16 together yield a proof of Lemma 3.6.

3.4. Time complexity. In this section, we describe two implementations of the online median algorithm given in section 3.1. Throughout this section, let ℓ denote the quantity $\log \frac{\Delta}{\delta}$. The first implementation runs in $O((n+\ell) \cdot n \log n)$ time. The second implementation runs in $O(n^2 + \ell n)$ time and assumes an $O(n^2)$-time preprocessing phase in which all distances are rounded down to the nearest integral power of λ. To analyze the running time of the implementations given below, we make use of the following lemma.

LEMMA 3.17. *Let $A = (x, r)$ be a child of a ball B in sequence σ_i, and let $A' = (x, r')$ be a child of a ball B' in sequence σ_j. If $i < j$, then $r \geq (\alpha + 1 + \frac{1}{\beta})r'$.*

Proof. We first obtain an upper bound on $d(x, z_i)$ by applying the λ-approximate triangle inequality to a sequence of points consisting of the centers of the balls in the suffix of σ_i beginning with ball A. Thus $d(x, z_i) \leq \lambda\beta\,(r + r/\alpha + \cdots) \leq \lambda\alpha\beta r/(\alpha - 1)$. By Lemma 3.7 and since $j > i$, we get that $\gamma r' \leq d(x, Z_j) \leq d(x, z_i)$. Combining these inequalities and using (4), we obtain

$$\begin{aligned} r &\geq \frac{(\alpha - 1)\gamma}{\lambda\alpha\beta} \cdot r' \\ &\geq \frac{\alpha - 1}{\alpha\beta} \cdot \left(\frac{\alpha^2\beta + \alpha\beta}{\alpha - 1} + \alpha \right) \lambda \cdot r' \\ &= \left(\alpha + 1 + \frac{1}{\beta} \right) r'. \qquad \square \end{aligned}$$

In the first implementation, for each point x in U, we sort the remaining points by their distance from x. The total sorting time is $O(n^2 \log n)$. Using these sorted arrays, we can compute the value of any given ball in $O(\log n)$ time. We also maintain the distance from x to the nearest point in Z_i. Note that $d(x, Z_{i+1})$ can be determined in constant time given $d(x, Z_i)$ and z_i. The total time to maintain such distances is thus $O(n^2)$. It follows that the first step of each iteration can be implemented in $O(n)$ time. The total time for the second step is $O(\log n)$ times the sum over all balls A appearing in some sequence σ_i, $0 \leq i < n$, of the number of children of A. By Lemma 3.17, it is straightforward to see that the latter sum is $O(\ell n)$, and thus the total time for the second step is $O(\ell n \log n)$. The running time of the third step is negligible. Thus the running time of the first implementation is $O((n + \ell) \cdot n \log n)$, as claimed above.

For the second implementation, note that after the preprocessing phase, there are $O(\ell)$ distinct distances. Thus, for each point x, $O(n+\ell)$ time is sufficient to construct

an $O(\ell)$-sized table that can be used to compute the value of any ball (x, r) in $O(1)$ time. It follows that the total time for the second step can be improved to $O(\ell n)$. The running time of the second implementation is therefore $O(n^2 + \ell n)$, which is linear in the size of the input (in bits).

4. Weakly λ-approximate metrics. The analysis in section 3 of this paper assumes that the (nonnegative, symmetric) distance function d approximately satisfies the triangle inequality. Recall that we defined a "λ-approximate" triangle inequality for $\lambda \geq 1$ as follows: For any sequence of points $x_0, \ldots, x_m$ in U, $d(x_0, x_m) \leq \lambda \cdot \sum_{0 \leq i < m} d(x_i, x_{i+1})$. We refer to such a distance function as a *λ-approximate metric*.

In this section, we show that the analysis in both sections 2 and 3 holds to within constant factors for an even weaker form of the triangle inequality. We say that a distance function d satisfies a "weakly λ-approximate" triangle inequality if, for any x, y, and z, $d(x, z) \leq \lambda(d(x, y) + d(y, z))$. We note that this inequality has been studied previously and is also referred to as the relaxed triangle inequality [10], the parameterized triangle inequality [1], and the λ-triangle inequality [9]. We will say that a distance function satisfying this inequality is a *weakly λ-approximate metric*. We will make use of such distance functions to extend our results to other objective functions. For example, the well-known k-means heuristic [8] has a sum of squared distances in its objective function. It is straightforward to show that squaring the distances in a metric yields a weakly 2-approximate metric. Thus the results in this section show that our analysis also holds, to within constant factors, with respect to the k-means objective function. (Remark: More generally, it is not hard to show that raising the distances in a metric to any constant power yields a weakly $O(1)$-approximate metric.)

Lemmas 4.1 and 4.2 establish that the approximation results in this paper hold, up to constant factors, even for weakly λ-approximate metrics. Recall that, in sections 2 and 3, we make use of the triangle inequality and the λ-approximate triangle inequality on sequences of points to derive upper bounds on the distances between pairs of points. In most cases, we consider constant-length sequences of points to derive our upper bounds. In such cases, Lemma 4.1 shows that a weakly λ-approximate metric is sufficient to guarantee that our upper bounds hold to within constant factors. Unfortunately, Lemma 4.1 alone is not sufficient to generalize our upper bounds based on nonconstant-length sequences of points, which arise in Lemmas 3.8, 3.16, and 3.17. For these cases, we require Lemma 4.2. Lemmas 4.1 and 4.2 together show that the upper bounds derived in Lemmas 3.8, 3.16, and 3.17 still hold up to constant factors given only a weakly λ-approximate triangle inequality.

LEMMA 4.1. *Let d be a weakly λ-approximate metric, and let $x_0, x_1, \ldots, x_m$ be points with $m \geq 1$. Then $d(x_0, x_m) \leq \lambda^{\lceil \log_2 m \rceil} \cdot \sum_{0 \leq i < m} d(x_i, x_{i+1})$.*

Proof. We will prove the lemma by induction. The base case, $m = 1$, is trivial. For the induction step, assume that for any sequence of points $y_0, \ldots, y_i$, $1 \leq i < m$, $d(y_0, y_i) \leq \lambda^{\lceil \log_2 i \rceil} \sum_{0 \leq j < i} d(y_j, y_{j+1})$. Then

$$
\begin{aligned}
d(x_0, x_m) &\leq \lambda \left(d(x_0, x_{\lceil \frac{m}{2} \rceil}) + d(x_{\lceil \frac{m}{2} \rceil}, x_m) \right) \\
&\leq \lambda \left(\lambda^{\lceil \log_2 \lceil \frac{m}{2} \rceil \rceil} \left(\sum_{0 \leq j < \lceil \frac{m}{2} \rceil} d(x_j, x_{j+1}) \right) + \lambda^{\lceil \log_2 \lfloor \frac{m}{2} \rfloor \rceil} \sum_{\lceil \frac{m}{2} \rceil \leq j < m} d(x_j, x_{j+1}) \right) \\
&\leq \lambda \cdot \lambda^{\lceil \log_2 m \rceil - 1} \sum_{0 \leq j < m} d(x_j, x_{j+1})
\end{aligned}
$$

$$= \lambda^{\lceil \log_2 m \rceil} \sum_{0 \le j < m} d(x_j, x_{j+1}).$$

The first step follows from the weakly λ-approximate triangle inequality. The second step follows by applying the induction hypothesis twice. (Note that $m \ge 2$ implies that $0 < \lceil \frac{m}{2} \rceil < m$, so the induction hypothesis is applicable.) The last step follows from the fact that $\lceil \log_2 \lceil \frac{m}{2} \rceil \rceil = \lceil \log_2 m \rceil - 1$. □

If λ and m are constant, then Lemma 4.1 implies that $d(x_0, x_m)$ is

$$\Theta\left(\sum_{0 \le i < m} d(x_i, x_{i+1}) \right).$$

Thus Lemma 4.1 is sufficient to show that the upper bounds derived in section 2 using the triangle inequality hold to within a constant factor given only a weakly λ-approximate metric. Similarly, the upper bounds derived in section 3 using the λ-approximate triangle inequality on constant-length sequences of points also hold to within constant factors given only a weakly λ-approximate metric. However, in Lemmas 3.8, 3.16, and 3.17, we derive upper bounds on distances by applying the λ-approximate triangle inequality to nonconstant-length sequences of points that appear in the sequences σ_i associated with our online median algorithm. In these cases, the nonconstant-length sequences of points we consider have the property that they are composed of a constant number of contiguous subsequences in which distances between successive points are either geometrically increasing or geometrically decreasing. Lemma 4.2 shows that the upper bounds derived using these sequences hold to within a constant factor assuming only a weakly λ-approximate metric.

LEMMA 4.2. *Let d be a weakly λ-approximate metric, and let $x_0, x_1, \ldots, x_m$ be points such that for $1 \le i \le m$, $d(x_i, x_{i+1}) \le d(x_{i-1}, x_i)/\xi$ for a positive real $\xi > \lambda$. Then $d(x_0, x_m) \le \frac{\lambda\xi}{\xi - \lambda} d(x_0, x_1)$.*

Proof. We first prove by induction that $d(x_0, x_m) \le \sum_{0 \le i < m} \lambda^{i+1} d(x_i, x_{i+1})$. For the base case, take $m = 1$. Then $d(x_0, x_1) \le \lambda d(x_0, x_1)$ since $\lambda \ge 1$. For the induction step, assume that for any sequence of points $y_0, \ldots, y_i$, $1 \le i < m$, $d(y_0, y_i) \le \sum_{0 \le j < i} \lambda^{j+1} d(y_j, y_{j+1})$. Observe that

$$\begin{aligned} d(x_0, x_m) &\le \lambda \left(d(x_0, x_1) + d(x_1, x_m) \right) \\ &\le \lambda d(x_0, x_1) + \lambda \left(\sum_{1 \le i < m} \lambda^i d(x_i, x_{i+1}) \right) \\ &\le \sum_{0 \le i < m} \lambda^{i+1} d(x_i, x_{i+1}), \end{aligned}$$

where the first step follows from the weakly λ-approximate triangle inequality and the second step follows from the induction hypothesis. Then

$$\begin{aligned} d(x_0, x_m) &\le \sum_{0 \le i < m} \lambda^{i+1} d(x_i, x_{i+1}) \\ &\le \sum_{0 \le i < m} \frac{\lambda^{i+1}}{\xi^i} d(x_0, x_1) \\ &\le \frac{\xi\lambda}{\xi - \lambda} d(x_0, x_1), \end{aligned}$$

where the second step follows from the assumption that $d(x_i, x_{i+1}) \leq d(x_{i-1}, x_i)/\xi$ for $0 \leq i < m$ and the third step follows from the assumption that $\xi > \lambda$. □

As stated above, Lemma 4.2 is needed in addition to Lemma 4.1 to show that the upper bounds derived in Lemmas 3.8, 3.16, and 3.17 hold to within a constant factor given only a weakly λ-approximate metric. We now explain how Lemmas 4.1 and 4.2 may be used to show that the upper bound obtained in part (i) of Lemma 3.8 holds to within a constant factor given a weakly λ-approximate metric. Recall that in part (i) of Lemma 3.8, we derive an upper bound on the distance $d(y', z_i)$. For the argument, we apply the λ-approximate triangle inequality to the sequence of points $\langle y', \ldots, y, x, \ldots, z_i \rangle$ and show that $d(y', z_i)$ is within a constant factor of the sum of the distances between successive points in this sequence. The prefix $\langle y', \ldots, y \rangle$ of this sequence appears in the sequence of balls σ_j associated with our online median algorithm. By the definition of our online median algorithm, the distances between successive points in $\langle y', \ldots, y \rangle$ decrease by a factor of β. Since β and λ are constants, and since $\beta > \lambda$, we can apply Lemma 4.2 with $\xi = \beta$ to conclude that $d(y', y)$ is within a constant factor of the sum of distances between successive points in $\langle y', \ldots, y \rangle$ given only a weakly λ-approximate metric. By a similar application of Lemma 4.2 to $d(x, z_i)$ with $\langle x, \ldots, z_i \rangle$ as the sequence of points, we can conclude that $d(x, z_i)$ is within a constant factor of the sum of distances between successive points in $\langle x, \ldots, z_i \rangle$ given only a weakly λ-approximate metric. With upper bounds on $d(y', y)$ and $d(x, z_i)$, we can then apply Lemma 4.1 to the constant-length sequence $\langle y', y, x, z_i \rangle$ to conclude that, given only a weakly λ-approximate metric, $d(y', z_i)$ is within a constant factor of the sum of distances between successive points in the sequence $\langle y', \ldots, y, x, \ldots, z_i \rangle$. Using Lemmas 4.1 and 4.2 in this manner, the bounds derived in part (iii) of Lemma 3.8 and in Lemmas 3.16 and 3.17 can also be shown to hold to within constant factors given only a weakly λ-approximate metric.

5. Concluding remarks. We plan to investigate whether the ideas presented in this paper can be applied to other problems. Korupolu, Plaxton, and Rajaraman [19] give an algorithm and an efficient distributed implementation for hierarchical cooperative caching in which the distance function is an ultrametric. We would like to see if the hierarchical greedy strategy can be used or extended to solve the cooperative caching problem in an arbitrary metric space. It would also be interesting to see if the hierarchical greedy strategy admits an efficient distributed implementation for this problem.

This paper has focused on the development of fast deterministic algorithms for the facility location problem and the online median problem. It is worth noting that there have been a number of recent results that make use of randomization to obtain fast algorithms for the k-median problem. The first such result was due to Indyk [14]; for the uniform-demand k-median problem, he gives a bicriteria approximation algorithm that uses random sampling and a black-box k-median algorithm. His algorithm has a constant probability of success and runs in $\tilde{O}(nk^3)$ time. (The $\tilde{O}$-tilde notation omits polylogarithmic factors in n and k.) Assuming the existence of an $\tilde{O}(n^2)$-time bicriteria k-median algorithm, this time bound can be reduced to $\tilde{O}(nk)$. Subsequently, Guha et al. obtained an $\tilde{O}(nk)$-time constant-factor approximation algorithm for the k-median problem in the data stream model of computation [13]. More recently, Thorup [27] has obtained a randomized constant-factor approximation algorithm for the k-median problem in a graph setting. For this problem, the interpoint distances are given by a graph on m edges rather than being fully specified in the input. That is, to obtain the distance between two points x and y, we must compute the shortest path

between x and y. Thorup gives an $\tilde{O}(m)$ constant-factor approximation algorithm for this problem. His algorithm implies an $\tilde{O}(nk)$-time algorithm for the version of the k-median problem defined in section 1.

Recently, we have obtained a randomized constant-factor approximation algorithm for the k-median problem that runs in $O(n(k + \log n) + k^2 \log^2 n)$ time under the standard assumption that the point weights and interpoint distances are polynomially bounded [24]. Thus, for k such that $\log n \leq k \leq n/\log^2 n$, our algorithm runs in $O(nk)$ time. Our algorithm succeeds *with high probability*, that is, for any positive constant ξ, we can adjust constant factors in the definition of the algorithm to achieve a failure probability less than $n^{-\xi}$. We also establish a matching $\Omega(nk)$ lower bound on the running time of any randomized constant-factor approximation algorithm for the k-median problem that has even a nonnegligible success probability (e.g., at least $\frac{1}{100}$).

Acknowledgments. The authors would like to thank the anonymous referees for making many helpful comments on the presentation.

REFERENCES

[1] T. Andreae and H.-J. Bandelt, *Performance guarantees for approximation algorithms depending on parametrized triangle inequalities*, SIAM J. Discrete Math., 8 (1995), pp. 1–16.

[2] V. Arya, N. Garg, R. Kandhekar, V. Pandit, A. Meyerson, and K. Munagala, *Local search heuristics for k-median and facility location problems*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, ACM, New York, 2001, pp. 21–29.

[3] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998.

[4] M. Charikar and S. Guha, *Improved combinatorial algorithms for facility location and k-median problems*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 378–388.

[5] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys, *A constant-factor approximation algorithm for the k-median problem*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 1–10.

[6] F. A. Chudak, *Improved approximation algorithms for uncapacitated facility location*, in Integer Programming and Combinatorial Optimization, R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, eds., Lecture Notes in Comput. Sci. 1412, Springer-Verlag, Berlin, 1998, pp. 180–194.

[7] W. Domschke and A. Drexl, *Location and Layout Planning: An International Bibliography*, Lecture Notes in Econom. and Math. Systems 238, Springer-Verlag, Berlin, 1985.

[8] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley, New York, 1973.

[9] R. Fagin, R. Kumar, and D. Sivakumar, *Computing top k lists*, in Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2003.

[10] R. Fagin and L. Stockmeyer, *Relaxing the triangle inequality in pattern matching*, Int. J. Comput. Vision, 30 (1998), pp. 219–231.

[11] M. X. Goemans and D. P. Williamson, *The primal-dual method for approximation algorithms and its application to network design problems*, in Approximation Algorithms for NP-Hard Problems, D. S. Hochbaum, ed., PWS, Boston, MA, 1995, pp. 144–191.

[12] S. Guha and S. Khuller, *Greedy strikes back: Improved facility location algorithms*, J. Algorithms, 31 (1999), pp. 228–248.

[13] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, *Clustering data streams*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2000, pp. 359–366.

[14] P. Indyk, *Sublinear time algorithms for metric space problems*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, ACM, New York, 1999, pp. 428–434.

[15] S. Irani and A. R. Karlin, *Online computation*, in Approximation Algorithms for NP-Hard Problems, D. S. Hochbaum, ed., PWS, Boston, MA, 1995, pp. 521–564.

[16] K. Jain, M. Mahdian, and A. Saberi, *A new greedy approach for facility location problems*, in Proceedings of the 34th ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 731–740.

[17] K. Jain and V. V. Vazirani, *Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation*, J. ACM, 48 (2001), pp. 274–296.

[18] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, *Analysis of a local search heuristic for facility location problems*, J. Algorithms, 37 (2000), pp. 146–188.

[19] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, *Placement algorithms for hierarchical cooperative caching*, J. Algorithms, 38 (2001), pp. 260–302.

[20] A. A. Kuehn and M. J. Hamburger, *A heuristic program for locating warehouses*, Management Sci., 9 (1963), pp. 643–666.

[21] J.-H. Lin and J. S. Vitter, *Approximation algorithms for geometric median problems*, Inform. Process. Lett., 44 (1992), pp. 245–249.

[22] J.-H. Lin and J. S. Vitter, *ε-approximations with minimum packing constraint violation*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 771–782.

[23] M. Mahdian, Y. Ye, and J. Zhang, *Improved approximation algorithms for metric facility location problems*, in Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization, K. Jansen, S. Leonardi, and V. Vazirani, eds., Lecture Notes in Comput. Sci. 2462, Springer-Verlag, Berlin, 2002, pp. 229–242.

[24] R. R. Mettu and C. G. Plaxton, *Optimal time bounds for approximate clustering*, in Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence, University of Alberta, Edmonton, Canada, 2002, pp. 344–351.

[25] P. Mirchandani and R. Francis, eds., *Discrete Location Theory*, John Wiley, New York, 1990.

[26] D. B. Shmoys, E. Tardos, and K. Aardal, *Approximation algorithms for facility location problems*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 265–274.

[27] M. Thorup, *Quick k-median, k-center, and facility location for sparse graphs*, in Proceedings of the 28th International Colloquium on Automata, Languages, and Programming, The Computer Technology Institute, Crete, Greece, 2001, pp. 249–260.

[28] N. E. Young, *K-medians, facility location, and the Chernoff-Wald bound*, in Proceedings of the 11th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2000, pp. 86–95.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 833–840

A FASTER SCALING ALGORITHM FOR MINIMIZING SUBMODULAR FUNCTIONS*

SATORU IWATA†

Abstract. Combinatorial strongly polynomial algorithms for minimizing submodular functions have been developed by Iwata, Fleischer, and Fujishige (IFF) and by Schrijver. The IFF algorithm employs a scaling scheme for submodular functions, whereas Schrijver's algorithm achieves strongly polynomial bound with the aid of distance labeling. Subsequently, Fleischer and Iwata have described a push/relabel version of Schrijver's algorithm to improve its time complexity. This paper combines the scaling scheme with the push/relabel framework to yield a faster combinatorial algorithm for submodular function minimization. The resulting algorithm improves over the previously best known bound by essentially a linear factor in the size of the underlying ground set.

Key words. submodular function, discrete optimization, algorithm

AMS subject classification. 90C27

DOI. 10.1137/S0097539701397813

1. Introduction.

Let V be a finite nonempty set of cardinality n. A set function f on V is *submodular* if it satisfies

$$f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y) \qquad \forall X, Y \subseteq V.$$

Submodular functions are discrete analogues of convex functions [14]. Examples of submodular functions include cut capacity functions, matroid rank functions, and entropy functions.

The first polynomial-time algorithm for submodular function minimization is due to Grötschel, Lovász, and Schrijver [9]. A strongly polynomial algorithm has also been described by Grötschel, Lovász, and Schrijver [10]. These algorithms rely on the ellipsoid method, which is not efficient in practice.

Recently, combinatorial strongly polynomial algorithms have been developed by Iwata, Fleischer, and Fujishige (IFF) [13] and by Schrijver [16]. Both of these algorithms build on works of Cunningham [2, 3]. The IFF algorithm employs a scaling scheme developed in capacity scaling algorithms for the submodular flow problem [7, 11]. In contrast, Schrijver [16] directly achieves a strongly polynomial bound by introducing a novel subroutine in the framework of lexicographic augmentation. Subsequently, Fleischer and Iwata [5, 6] have described a push/relabel algorithm using Schrijver's subroutine to improve the running time bound. In this paper, we combine the scaling scheme with the push/relabel technique to yield a faster combinatorial algorithm.

Let γ denote the time required for computing the function value of f and M denote the maximum absolute value of f. The IFF scaling algorithm minimizes an integral submodular function in $O(n^5\gamma \log M)$ time. The strongly polynomial version

*Received by the editors November 12, 2001; accepted for publication (in revised form) March 31, 2003; published electronically June 10, 2003. This research was supported in part by the Sumitomo Foundation and a Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan.

http://www.siam.org/journals/sicomp/32-4/39781.html

†Department of Mathematical Informatics, University of Tokyo, Tokyo 113-8656, Japan (iwata@mist.i.u-tokyo.ac.jp).

runs in $O(n^7\gamma \log n)$ time, whereas an improved variant of Schrijver's algorithm runs in $O(n^7\gamma + n^8)$ time [6].

The time complexity of our new scaling algorithm is $O((n^4\gamma + n^5)\log M)$. Since the function evaluation oracle has to identify an arbitrary subset of V as its argument, it is natural to assume γ is at least linear in n. With this assumption, the new algorithm is faster than the IFF algorithm by a factor of n. The strongly polynomial version of the new scaling algorithm runs in $O((n^6\gamma + n^7)\log n)$ time. This is an improvement over the previous best bound by essentially a linear factor in n.

These combinatorial algorithms perform multiplications and divisions, although the problem of submodular function minimization does not involve those operations. Schrijver [16] asks if one can minimize submodular functions in strongly polynomial time using only additions, subtractions, comparisons, and oracle calls for the function values. Such an algorithm is called "fully combinatorial." A very recent paper [12] settles this problem by developing a fully combinatorial variant of the IFF algorithm. Similarly, we can implement the strongly polynomial version of our scaling algorithm in a fully combinatorial manner. The resulting algorithm runs in $O(n^8\gamma \log^2 n)$ time, improving the previous $O(n^9\gamma \log^2 n)$ bound by a factor of n.

This paper is organized as follows. Section 2 provides preliminaries on submodular functions. In section 3, we describe the new scaling algorithm. Section 4 is devoted to its complexity analysis. Finally, in section 5, we discuss its extensions as well as a fully combinatorial implementation.

2. Preliminary. This section provides preliminaries on submodular functions. See [8, 14] for more details and general background.

For a vector $x \in \mathbf{R}^V$ and a subset $Y \subseteq V$, we denote $x(Y) = \sum_{u\in Y} x(u)$. We also denote x^- the vector in $\mathbf{R}^V$ with $x^-(u) = \min\{x(u), 0\}$. For each $u \in V$, let χ_u denote the vector in $\mathbf{R}^V$ with $\chi_u(u) = 1$ and $\chi_u(v) = 0$ for $v \in V\backslash\{u\}$.

For a submodular function $f : 2^V \to \mathbf{R}$ with $f(\emptyset) = 0$, we consider the *base polyhedron*

$$\mathrm{B}(f) = \{x \mid x \in \mathbf{R}^V,\ x(V) = f(V),\ \forall Y \subseteq V : x(Y) \leq f(Y)\}.$$

A vector in $\mathrm{B}(f)$ is called a *base*. In particular, an extreme point of $\mathrm{B}(f)$ is called an *extreme base*. An extreme base can be computed by the greedy algorithm of Edmonds [4] and Shapley [17] as follows.

Let $L = (v_1, \ldots, v_n)$ be a linear ordering of V. For any $v_j \in V$, we denote $L(v_j) = \{v_1, \ldots, v_j\}$. The greedy algorithm with respect to L generates an extreme base $y \in \mathrm{B}(f)$ by

$$y(u) := f(L(u)) - f(L(u)\backslash\{u\}). \tag{2.1}$$

Conversely, any extreme base can be obtained in this way with an appropriate linear ordering.

LEMMA 2.1. *Let Q and R be disjoint subsets of V such that $Q \cup R$ forms an interval in L. Let L' be the linear ordering obtained from L by moving Q to the place immediately after R without changing the orderings in Q and in R. Then the extreme base y' generated by L' satisfies $y'(q) \leq y(q)$ for $q \in Q$ and $y'(r) \geq y(r)$ for $r \in R$.*

Proof. For any $q \in Q$, we have $L'(q) \supseteq L(q)$. Therefore, the submodularity of f implies $y'(q) = f(L'(q)) - f(L'(q)\backslash\{q\}) \leq f(L(q)) - f(L(q)\backslash\{q\}) = y(q)$. Similarly, $L'(r) \subseteq L(r)$ holds for $r \in R$. Then it follows from the submodularity of f that $y'(r) = f(L'(r)) - f(L'(r)\backslash\{r\}) \geq f(L(r)) - f(L(r)\backslash\{r\}) = y(r)$. ∎

For any base $x \in \mathrm{B}(f)$ and any subset $Y \subseteq V$, we have $x^-(V) \leq x(Y) \leq f(Y)$. The following theorem shows that these inequalities are in fact tight for appropriately chosen x and Y.

THEOREM 2.2. *For a submodular function $f : 2^V \to \mathbf{R}$, we have*

$$\max\{x^-(V) \mid x \in \mathrm{B}(f)\} = \min\{f(Y) \mid Y \subseteq V\}.$$

Moreover, if f is integer-valued, then the maximizer x can be chosen from among integral bases. □

This theorem is immediate from the vector reduction theorem on polymatroids due to Edmonds [4]. It has motivated combinatorial algorithms for minimizing submodular functions.

3. A scaling algorithm. This section presents a new scaling algorithm for minimizing an integral submodular function $f : 2^V \to \mathbf{Z}$.

The algorithm consists of scaling phases with a scale parameter $\delta \geq 0$. It keeps a set of linear orderings $\{L_i \mid i \in I\}$ of the vertices in V. We denote $v \preceq_i u$ if v precedes u in L_i or $v = u$. Each linear ordering L_i generates an extreme base $y_i \in \mathrm{B}(f)$ by the greedy algorithm. The algorithm also keeps a base $x \in \mathrm{B}(f)$ as a convex combination $x = \sum_{i \in I} \lambda_i y_i$ of the extreme bases. Initially, $I = \{0\}$ with an arbitrary linear ordering L_0 and $\lambda_0 = 1$.

Furthermore, the algorithm works with a flow in the complete directed graph on the vertex set V. The flow is represented as a skew-symmetric function $\varphi : V \times V \to \mathbf{R}$. Each arc capacity is equal to δ. Namely, $\varphi(u, v) + \varphi(v, u) = 0$ and $-\delta \leq \varphi(u, v) \leq \delta$ hold for any pair of vertices $u, v \in V$. The boundary $\partial\varphi$ is defined by $\partial\varphi(u) = \sum_{v \in V} \varphi(u, v)$ for $u \in V$. Initially, $\varphi(u, v) = 0$ for any $u, v \in V$.

Each scaling phase aims at increasing $z^-(V)$ for $z = x + \partial\varphi$. Given a flow φ, the procedure constructs an auxiliary directed graph $G_\varphi = (V, A_\varphi)$ with arc set $A_\varphi = \{(u, v) \mid u \neq v,\ \varphi(u, v) \leq 0\}$. Let $S = \{v \mid z(v) \leq -\delta\}$ and $T = \{v \mid z(v) \geq \delta\}$. A directed path in G_φ from S to T is called an *augmenting path*.

Each scaling phase also keeps a valid labeling d. A labeling $d : V \to \mathbf{Z}$ is *valid* if $d(u) = 0$ for $u \in S$ and $v \preceq_i u$ implies $d(v) \leq d(u) + 1$. A valid labeling $d(v)$ serves as a lower bound on the number of arcs from S to v in the directed graph $G_I = (V, A_I)$ with the arc set $A_I = \{(u, v) \mid \exists i \in I,\ v \preceq_i u\}$.

If there is an augmenting path P, the algorithm augments the flow φ along P by $\varphi(u, v) := \varphi(u, v) + \delta$ and $\varphi(v, u) := \varphi(v, u) - \delta$ for each arc (u, v) in P. This procedure is referred to as $\mathsf{Augment}(\varphi, P)$. As a result of $\mathsf{Augment}(\varphi, P)$, the initial vertex s of P may get rid of S and no new vertex joins S. Thus $\mathsf{Augment}(\varphi, P)$ does not violate the validity of d.

Let W be the set of vertices reachable from S in G_φ, and let Z be the set of vertices that attains the minimum labeling in $V \backslash W$. A pair (u, v) of $u \in W$ and $v \in Z$ is called *active* for $i \in I$ if v is the first vertex of Z in L_i and u is the last vertex in L_i with $v \preceq_i u$ and $d(v) = d(u) + 1$. A triple (i, u, v) is also called *active* if (u, v) is active for $i \in I$. The procedure $\mathsf{Multiple\text{-}Exchange}(i, u, v)$ is applicable to an active triple (i, u, v).

For an active triple (i, u, v), the set of vertices from v to u in L_i is called an *active interval*. The active interval is divided into Q and R by $Q = \{w \mid w \in W,\ v \prec_i w \preceq_i u\}$ and $R = \{w \mid w \in V \backslash W,\ v \preceq_i w \prec_i u\}$.

The procedure $\mathsf{Multiple\text{-}Exchange}(i, u, v)$ moves the vertices in R to the place immediately after u in L_i, without changing the ordering in Q and in R. Then it computes an extreme base y_i generated by the new L_i. By Lemma 2.1, this results

in $y_i(q) \geq y_i^\circ(q)$ for $q \in Q$ and $y_i(r) \leq y_i^\circ(r)$ for $r \in R$, where y_i° denotes the previous y_i.

Consider a complete bipartite graph with the vertex sets Q and R. The algorithm finds a flow $\xi : Q \times R \to \mathbf{R}_+$ such that $\sum_{r \in R} \xi(q,r) = y_i(q) - y_i^\circ(q)$ for each $q \in Q$ and $\sum_{q \in Q} \xi(q,r) = y_i^\circ(r) - y_i(r)$ for each $r \in R$. Such a flow can be obtained easily by the so-called northwest corner rule. Then the procedure computes $\alpha = \min\{\lambda_i, \delta/\beta\}$ with $\beta = \max\{\xi(q,r) \mid q \in Q,\, r \in R\}$ and moves x by $x := x + \alpha(y_i - y_i^\circ)$. In order to keep z invariant, the procedure adjusts the flow φ by $\varphi(q,r) := \varphi(q,r) - \alpha\xi(q,r)$ and $\varphi(r,q) := \varphi(r,q) + \alpha\xi(q,r)$ for every $(q,r) \in Q \times R$. The resulting φ satisfies the capacity constraints due to the choice of α, and the vertices in W remain reachable from S in G_φ.

If $\alpha = \lambda_i$, Multiple-Exchange(i,u,v) is called *saturating*. Otherwise, it is called *nonsaturating*. In a nonsaturating Multiple-Exchange(i,u,v), a new index k is added to I. The associated linear ordering L_k is the previous L_i. The coefficient λ_k is determined by $\lambda_k := \lambda_i - \alpha$, and then λ_i is replaced by $\lambda_i := \alpha$. Thus the algorithm continues to keep x as a convex combination $x = \sum_{i \in I} \lambda_i y_i$.

Suppose the labeling d is valid before the algorithm applies Multiple-Exchange to an active triple (i,u,v). For any vertex w in the active interval, $d(v) \leq d(w) + 1$ and $d(w) \leq d(u)+1$ hold. These inequalities and $d(v) = d(u)+1$ imply $d(v) \leq d(w) \leq d(u)$. Note that $d(v) \leq d(r)$ holds for any $r \in R \subseteq V \backslash W$. Hence we have $d(r) = d(v)$ for any $r \in R$. If Multiple-Exchange(i,u,v) adds a new arc (s,t) to A_I, then $s \in Q$ and $t \in R$. Therefore, we have $d(t) = d(v) \leq d(s) + 1$. Thus Multiple-Exchange$(i,u,v)$ does not violate the validity of d.

Let h denote the number of vertices in the active interval. The number of function evaluations required for computing the new extreme base y_i by the greedy algorithm is at most h. The northwest corner rule can be implemented to run in $\mathrm{O}(h)$ time, and the number of arcs (q,r) with $\xi(q,r) > 0$ is at most $h - 1$. Thus the total time complexity of Multiple-Exchange(i,u,v) is $\mathrm{O}(h\gamma)$.

If there is no active triple, the algorithm applies Relabel to each $v \in Z$. The procedure Relabel(v) increments $d(v)$ by one. Then the labeling d remains valid.

The number of extreme bases in the expression of x increases by one as a result of nonsaturating Multiple-Exchange. In order to reduce the complexity, the algorithm occasionally applies a procedure Reduce(x,I) that computes an expression of x as a convex combination of affinely independent extreme bases chosen from the currently used ones. This computation takes $\mathrm{O}(n^2|I|)$ time with the aid of Gaussian elimination.

We are now ready to describe the new scaling algorithm.

Step 0: Let L_0 be an arbitrary linear ordering. Compute an extreme base y_0 by the greedy algorithm with respect to L_0. Put $x := y_0$, $\lambda_0 := 1$, $I := \{0\}$, and $\delta := |x^-(V)|/n^2$.

Step 1: Put $d(v) := 0$ for $v \in V$, and $\varphi(u,v) := 0$ for $u, v \in V$.

Step 2: Put $S := \{v \mid z(v) \leq -\delta\}$ and $T := \{v \mid z(v) \geq \delta\}$, where $z = x + \partial\varphi$. Let W be the set of vertices reachable from S in G_φ.

Step 3: If there is an augmenting path P, then do the following.

(3-1) Apply Augment(φ, P).

(3-2) Apply Reduce(x,I).

(3-3) Go to Step 2.

Step 4: Compute $\ell := \min\{d(v) \mid v \in V \backslash W\}$ and put $Z := \{v \mid v \in V \backslash W,\, d(v) = \ell\}$. If $\ell < n$, then do the following.

(4-1) If there is an active triple (i,u,v), then apply Multiple-Exchange(i,u,v).

(4-2) Otherwise, apply Relabel(v) for each $v \in Z$.

(4-3) Go to Step 2.

Step 5: Determine the set X of vertices reachable from S in G_I. If $\delta \geq 1/n^2$, then apply Reduce(x, I), $\delta := \delta/2$, and go to Step 1.

We now show that the last set X obtained by the scaling algorithm is a minimizer of f.

LEMMA 3.1. *At the end of each scaling phase,* $z^-(V) \geq f(X) - n(n+1)\delta/2$.

Proof. At the end of each scaling phase, $d(v) = n$ for every $v \in V \backslash W$. Since $d(v)$ is a lower bound on the number of arcs from S to v in G_I, this means there is no directed path from S to $V \backslash W$ in G_I. Thus we have $X \subseteq W \subseteq V \backslash T$, which implies $z(v) \leq \delta$ for $v \in X$. It follows from $S \subseteq X$ that $z(v) \geq -\delta$ for $v \in V \backslash X$. Since there is no arc in G_I emanating from X, we have $y_i(X) = f(X)$ for each $i \in I$, and hence $x(X) = \sum_{i \in I} \lambda_i y_i(X) = f(X)$. We also have $\partial\varphi(X) \geq -\delta\,|X| \cdot |V \backslash X| \geq -n(n-1)\delta/2$. Therefore, we have $z^-(V) = z^-(X) + z^-(V \backslash X) \geq z(X) - \delta|X| - \delta|V \backslash X| = x(X) + \partial\varphi(X) - n\delta \geq f(X) - n(n+1)\delta/2$. □

LEMMA 3.2. *At the end of each scaling phase,* $x^-(V) \geq f(X) - n^2\delta$.

Proof. The set $Y = \{v \mid x(v) < 0\}$ satisfies $x^-(V) = x(Y) = z(Y) - \partial\varphi(Y) \geq z^-(V) - \partial\varphi(Y)$. Note that $\partial\varphi(Y) \leq \delta\,|Y| \cdot |V \backslash Y| \leq n(n-1)\delta/2$. Therefore, we have $x^-(V) \geq z^-(V) - n(n-1)\delta/2$, which together with Lemma 3.1 implies $x^-(V) \geq f(X) - n^2\delta$. □

THEOREM 3.3. *At the end of the last scaling phase,* X *is a minimizer of* f.

Proof. Since $\delta < 1/n^2$ in the last scaling phase, Lemma 3.2 implies $x^-(V) > f(X) - 1$. Then it follows from the integrality of f that $f(X) \leq f(Y)$ holds for any $Y \subseteq V$. □

4. Complexity. This section is devoted to complexity analysis of the new scaling algorithm.

LEMMA 4.1. *Each scaling phase performs* Augment $O(n^2)$ *times.*

Proof. At the beginning of each scaling phase, the set X obtained in the previous scaling phase satisfies $z^-(V) \geq f(X) - 2n^2\delta$ by Lemma 3.2. For the first scaling phase, we have the same inequality by taking $X = \emptyset$. Note that $z^-(V) \leq z(X) \leq f(X) + n(n-1)\delta/2$ throughout the algorithm. Thus each scaling phase increases $z^-(V)$ by at most $3n^2\delta$. Since each augmentation increases $z^-(V)$ by δ, each scaling phase performs at most $3n^2$ augmentations. □

LEMMA 4.2. *Each scaling phase performs* Relabel $O(n^2)$ *times.*

Proof. Each application of Relabel(v) increases $d(v)$ by one. Since Relabel(v) is applied only if $d(v) < n$, Relabel(v) is applied at most n times for each $v \in V$ in a scaling phase. Thus the total number of relabels in a scaling phase is at most n^2. □

LEMMA 4.3. *The number of indices in* I *is at most* $2n$.

Proof. A new index is added as a result of nonsaturating Multiple-Exchange. In a nonsaturating Multiple-Exchange(i, u, v), the arc (q, r) that determines β satisfies $\varphi(q, r) \leq 0$ after the update of φ, and the vertex r in R becomes reachable from S in G_φ. This means the set W is enlarged. Thus there are at most n applications of nonsaturating Multiple-Exchange between augmentations. Hence the number of indices added between augmentations is at most n. After each augmentation, the number of indices is reduced to at most n. Therefore, $|I| \leq 2n$ holds throughout the algorithm. □

In order to analyze the number of function evaluations in each scaling phase, we now introduce the notion of reordering phase. A *reordering phase* consists of consecutive applications of Multiple-Exchange between those of Relabel or Reduce. By Lemmas 4.1 and 4.2, each scaling phase performs $O(n^2)$ reordering phases.

LEMMA 4.4. *There are* $O(n^2)$ *function evaluations in each reordering phase.*

Proof. The number of function evaluations in Multiple-Exchange(i, u, v) is at most the number of vertices in the active interval for (i, u, v). In order to bound the total number of function evaluations in a reordering phase, suppose the procedure Multiple-Exchange(i, u, v) marks each pair (i, w) for w in the active interval. We now intend to claim that any pair (i, w) of $i \in I$ and $w \in V$ is marked at most once in a reordering phase.

In a reordering phase, the algorithm does not change the labeling d nor does it delete a vertex from W. Hence the minimum value of d in $V\backslash W$ is nondecreasing. After execution of Multiple-Exchange(i, u, v), there will not be an active pair for i until the minimum value of d in $V\backslash W$ becomes larger. Let Multiple-Exchange(i, s, t) be the next application of Multiple-Exchange to the same index $i \in I$. Then we have $d(t) > d(v) = d(u) + 1$, which implies $v \prec_i u \prec_i t \prec_i s$ in the linear ordering L_i before Multiple-Exchange(i, u, v). Thus a pair (i, w) marked in Multiple-Exchange(i, u, v) will not be marked again in the reordering phase.

Since $|I| \leq 2n$ by Lemma 4.3, there are at most $2n^2$ possible marks without duplications. Therefore, the total number of function evaluations in a reordering phase is $O(n^2)$. □

In order to find an active triple efficiently in Step (4-1), we keep track of possible candidates of active triples. For each $i \in I$ and $\ell = 1, \ldots, n-1$, let $u_{i\ell}$ denote the last vertex u in L_i such that $u \in W$ and $d(u) = \ell - 1$. Similarly, $v_{i\ell}$ denotes the first vertex v in L_i such that $v \in V\backslash W$ and $d(v) = \ell$. Then $(i, u_{i\ell}, v_{i\ell})$ is an active triple if $\ell = \min\{d(v) \mid v \in V\backslash W\}$ and $v_{i\ell} \prec u_{i\ell}$. At the beginning of each reordering phase, we scan the linear orderings to find all those candidates in $O(n^2)$ time.

In the rest of the reordering phase, we update the candidates whenever a new vertex is added to W. Let w be the vertex that is added to W. For each $i \in I$, if $u_{i\ell} \preceq w$ with $d(w) = \ell - 1$, then we replace $u_{i\ell}$ by w. If $w = v_{i\ell}$, then we find the new $v_{i\ell}$ by scanning L_i. Thus it takes $O(n^2)$ time to update the candidates when a new vertex is added to W. Since at most n vertices are added to W, each reordering phase requires $O(n^3)$ fundamental operations.

THEOREM 4.5. *The algorithm performs* $O(n^4 \log M)$ *function evaluations and* $O(n^5 \log M)$ *arithmetic computations.*

Proof. Consider the set $U = \{u \mid x(u) > 0\}$ for the initial base $x \in B(f)$. Then we have $x^-(V) = x(V) - x(U) \geq f(V) - f(U) \geq -2M$. Therefore, the initial value of δ satisfies $\delta \leq 2M/n^2$. Each scaling phase cuts the value of δ in half, and the algorithm terminates when $\delta < 1/n^2$. Thus the algorithm consists of $O(\log M)$ scaling phases.

Since each scaling phase performs $O(n^2)$ reordering phases, Lemma 4.4 implies that the number of function evaluations in a scaling phase is $O(n^4)$. In addition, each reordering phase requires $O(n^3)$ steps to keep track of active triples. By Lemma 4.1, each scaling phase performs $O(n^2)$ calls of Reduce, which requires $O(n^3)$ arithmetic computations. Thus each scaling phase consists of $O(n^4)$ function evaluations and $O(n^5)$ arithmetic computations. Therefore, the total running time bound is $O((n^4\gamma + n^5) \log M)$. □

5. Discussions. A family $\mathcal{D} \subseteq 2^V$ is called a distributive lattice (or a ring family) if $X \cap Y \in \mathcal{D}$ and $X \cup Y \in \mathcal{D}$ for any pair of $X, Y \in \mathcal{D}$. A compact representation of $\mathcal{D}$ is given by a directed graph as follows. Let $D = (V, F)$ be a directed graph with the arc set F. A subset $Y \subseteq V$ is called an ideal of D if no arc enters Y in D. Then the set of ideals of D forms a distributive lattice. Conversely, any distributive lattice $\mathcal{D} \subseteq 2^V$ with $\emptyset, V \in \mathcal{D}$ can be represented in this way due to

Birkhoff's representation theorem [1, Theorem 2.5]. Moreover, contracting strongly connected components of D to single vertices, we may assume that the directed graph D is acyclic.

For minimizing a submodular function f on $\mathcal{D}$, we apply the scaling algorithm with a minor modification. The modified version uses the directed graph $G_\varphi = (V, A_\varphi \cup F)$ instead of $G_\varphi = (V, A_\varphi)$. The initial linear ordering L_0 must be consistent with D; i.e., $v \preceq_i u$ if $(u, v) \in F$. Then all the linear orderings that appear in the algorithm will be consistent with D. This ensures that the set X obtained at the end of each scaling phase belongs to $\mathcal{D}$. Thus the modification of our scaling algorithm finds a minimizer of f in $\mathcal{D}$.

Iwata, Fleischer, and Fujishige [13] also describe a strongly polynomial algorithm that repeatedly applies their scaling algorithm with $O(\log n)$ scaling phases. The number of iterations is $O(n^2)$. Replacing the scaling algorithm by the new one, we obtain an improved strongly polynomial algorithm that runs in $O((n^6\gamma + n^7)\log n)$ time.

A very recent paper [12] has shown that the strongly polynomial IFF algorithm can be implemented by using only additions, subtractions, comparisons, and oracle calls for function values. Similarly, the new strongly polynomial scaling algorithm can be made fully combinatorial as follows.

The first step towards a fully combinatorial implementation is to neglect Reduce. This causes growth of the number of extreme bases for convex combination. However, the number is still bounded by a polynomial in n. Since the number of indices added between augmentations is at most n, each scaling phase yields $O(n^3)$ new extreme bases. Hence the number of extreme bases through the $O(\log n)$ scaling phases is $O(n^3 \log n)$.

The next step is to choose an appropriate step length in Multiple-Exchange so that the coefficients should be rational numbers with a common denominator bounded by a polynomial in n. Let θ denote the value of δ in the first scaling phase. Then $\kappa = \theta/\delta$ is an integer. For each $i \in I$, we keep $\lambda_i = \mu_i/\kappa$ with an integer μ_i. We then modify the definition of saturating Multiple-Exchange. Multiple-Exchange(i, u, v) is now called saturating if $\lambda_i \xi(q, r) \le \varphi(q, r)$ for every $(q, r) \in Q \times R$. Otherwise, it is called nonsaturating. In nonsaturating Multiple-Exchange(i, u, v), let ν be the minimum integer such that $\nu\xi(q, r) > \varphi(q, r)\kappa$ for some $(q, r) \in Q \times R$. Such an integer ν can be computed by binary search. Then the new coefficients λ_k and λ_i are determined by $\mu_k := \mu_i - \nu$ and $\mu_i := \nu$. Thus the coefficients are rational numbers whose common denominator is κ, which is bounded by a polynomial in n through the $O(\log n)$ scaling phases. Then it is easy to implement this algorithm using only additions, subtractions, comparisons, and oracle calls for the function values.

Finally, we discuss time complexity of the resulting fully combinatorial algorithm. The algorithm performs $O(n^2)$ iterations of $O(\log n)$ scaling phases. Since it keeps $O(n^3 \log n)$ extreme bases, each scaling phase requires $O(n^6 \log n)$ oracle calls for function evaluations and $O(n^7 \log n)$ fundamental operations. Therefore, the total running time is $O((n^8\gamma + n^9)\log^2 n)$. This improves the previous $O(n^9\gamma \log^2 n)$ bound in [12] by essentially a linear factor in n.

In order to reduce this time complexity, McCormick [15] suggests a more efficient implementation for finding active triples. For each $i \in I$ and $\ell = 1, \ldots, n$, let $\sigma_{i\ell}$ denote the last vertex s in L_i with $d(s) = \ell - 1$. Similarly, $\tau_{i\ell}$ denotes the first vertex t in L_i with $d(t) = \ell$. Then there is an active triple (i, u, v) with $d(v) = \ell$ only if $\tau_{i\ell} \prec_i \sigma_{i\ell}$. At the beginning of each reordering phase we scan the linear orderings to

find all $\sigma_{i\ell}$ and $\tau_{i\ell}$ in $O(n|I|)$ time. Note that within the reordering phase, $\sigma_{i\ell}$ and $\tau_{i\ell}$ are invariant until the algorithm performs Multiple-Exchange(i, u, v) with $d(v) = \ell$. Once such a Multiple-Exchange is applied, there will be no active triples for the same i and ℓ in the rest of the reordering phase.

For a pair of i and ℓ with $\tau_{i\ell} \prec \sigma_{i\ell}$, we may restrict the search for active triples to the interval between $\tau_{i\ell}$ and $\sigma_{i\ell}$ in L_i. Since these intervals are disjoint, the total number of fundamental operations required for finding active triples is $O(n|I|)$ in each reordering phase. This reduces the number of fundamental operations in a scaling phase to $O(n^6 \log n)$. Thus the resulting fully combinatorial algorithm runs in $O(n^8 \gamma \log^2 n)$ time.

Acknowledgments. The author is grateful to Lisa Fleischer, Satoru Fujishige, Yasuko Matsui, Tom McCormick, and Kazuo Murota for stimulating conversations and helpful comments on the manuscript. The idea of Multiple-Exchange was originally suggested by Satoru Fujishige as heuristics to improve the practical performance of the IFF algorithm. Lisa Fleischer kindly pointed out an error in an earlier version of this paper, and Tom McCormick generously allowed me to include his idea for implementing the algorithm that leads to the bound of the fully combinatorial version. Finally, the author wishes to thank two anonymous referees for many helpful suggestions to improve the presentation of this paper.

REFERENCES

[1] M. Aigner, *Combinatorial Theory*, Springer-Verlag, Berlin, New York, 1979.
[2] W. H. Cunningham, *Testing membership in matroid polyhedra*, J. Combin. Theory Ser. B, 36 (1984), pp. 161–188.
[3] W. H. Cunningham, *On submodular function minimization*, Combinatorica, 5 (1985), pp. 185–192.
[4] J. Edmonds, *Submodular functions, matroids, and certain polyhedra*, in Combinatorial Structures and Their Applications, R. Guy, H. Hanani, N. Sauer, and J. Schönheim, eds., Gordon and Breach, New York, 1970, pp. 69–87.
[5] L. Fleischer and S. Iwata, *Improved algorithms for submodular function minimization and submodular flow*, in Proceedings of the 32nd ACM Symposium on Theory of Computing, Portland, OR, 2000, ACM, New York, 2000, pp. 107–116.
[6] L. Fleischer and S. Iwata, *A push-relabel framework for submodular function minimization and applications to parametric optimization*, Discrete Appl. Math., to appear.
[7] L. Fleischer, S. Iwata, and S. T. McCormick, *A faster capacity scaling algorithm for submodular flow*, Math. Program., 92 (2002), pp. 119–139.
[8] S. Fujishige, *Submodular Functions and Optimization*, North-Holland, Amsterdam, 1991.
[9] M. Grötschel, L. Lovász, and A. Schrijver, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, 1 (1981), pp. 169–197.
[10] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.
[11] S. Iwata, *A capacity scaling algorithm for convex cost submodular flows*, Math. Programming, 76 (1997), pp. 299–308.
[12] S. Iwata, *A fully combinatorial algorithm for submodular function minimization*, J. Combin. Theory Ser. B, 84 (2002), pp. 203–212.
[13] S. Iwata, L. Fleischer, and S. Fujishige, *A combinatorial strongly polynomial algorithm for minimizing submodular functions*, J. ACM, 48 (2001), pp. 761–777.
[14] L. Lovász, *Submodular functions and convexity*, in Mathematical Programming — The State of the Art, A. Bachem, M. Grötschel, and B. Korte, eds., Springer-Verlag, Berlin, 1983, pp. 235–257.
[15] S. T. McCormick, *Submodular function minimization*, in Handbook of Discrete Optimization, K. Aardal, G. Nemhauser, and R. Weismantel, eds., Elsevier, New York, to appear.
[16] A. Schrijver, *A combinatorial algorithm minimizing submodular functions in strongly polynomial time*, J. Combin. Theory Ser. B, 80 (2000), pp. 346–355.
[17] L. S. Shapley, *Cores of convex games*, Internat. J. Game Theory, 1 (1971), pp. 11–26.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 841–863

THE PARTITION TECHNIQUE FOR OVERLAYS OF ENVELOPES*

VLADLEN KOLTUN† AND MICHA SHARIR‡

Abstract. We obtain a near-tight bound of $O(n^{3+\varepsilon})$ for any $\varepsilon > 0$ on the complexity of the overlay of the minimization diagrams of two collections of surfaces in four dimensions. This settles a long-standing problem in the theory of arrangements, most recently cited by Agarwal and Sharir [in *Handbook of Computational Geometry*, North–Holland, Amsterdam, 2000, pp. 49–119, Open Problem 2], and substantially improves and simplifies a result previously published by the authors [in *Proceedings of the* 13*th ACM–SIAM Symposium on Discrete Algorithms*, ACM, New York, SIAM, Philadelphia, 2002, pp. 810–819].

Our bound is obtained by introducing a new approach to the analysis of combinatorial structures arising in geometric arrangements of surfaces. This approach, which we call the "partition technique," is based on k-fold divide and conquer, in which a given collection $\mathcal{F}$ of n surfaces is partitioned into k subcollections $\mathcal{F}_i$ of n/k surfaces each, and the complexity of the relevant combinatorial structure in $\mathcal{F}$ is recursively related to the complexities of the corresponding structures in each of the $\mathcal{F}_i$'s. We introduce this approach by applying it first to obtain a new *simple* proof for the known near-quadratic bound on the complexity of an overlay of two minimization diagrams of collections of surfaces in $\mathbb{R}^3$, thereby simplifying the previously available proof [P. K. Agarwal, O. Schwarzkopf, and M. Sharir, *Discrete Comput. Geom.*, 15 (1996), pp. 1–13].

The main new bound on overlays has numerous algorithmic and combinatorial applications, some of which are presented in this paper.

Key words. computational geometry, arrangement, envelope, overlay, partition technique

AMS subject classifications. 68U05, 52C45, 68Q25

DOI. 10.1137/S009753970240700X

1. Introduction.

In this paper, we obtain combinatorial bounds on overlays of minimization diagrams by introducing a new approach to the analysis of combinatorial structures in arrangements of surfaces. Let us start with the basic definitions. (For a thorough treatment of the topic we are about to briefly introduce, the reader is referred to [21, Chapter 7].)

Let $\mathcal{F}$ be a family of n d-variate (not necessarily continuous or totally defined) functions of *constant description complexity*; that is, the graph of each function is a semialgebraic set in $\mathbb{R}^{d+1}$ defined by a constant number of polynomial equalities and inequalities of constant maximum degree. The lower envelope $E_{\mathcal{F}}$ of $\mathcal{F}$ is the pointwise minimum of the functions of $\mathcal{F}$:

$$E_{\mathcal{F}}(\mathbf{x}) = \min_{f \in \mathcal{F}} f(\mathbf{x}) \qquad \text{for } \mathbf{x} \in \mathbb{R}^d.$$

*Received by the editors May 6, 2002; accepted for publication (in revised form) February 24, 2003; published electronically June 10, 2003. This work was supported by a grant from the Israel Science Fund (for a Center of Excellence in Geometric Computing) and is part of the first author's Ph.D. dissertation, prepared under the supervision of the second author at Tel Aviv University. A preliminary version of this paper appeared in the *Proceedings of the* 43*rd IEEE Symposium on Foundations of Computer Science*, Vancouver, Canada, 2002.

http://www.siam.org/journals/sicomp/32-4/40700.html

†Computer Science Division, University of California Berkeley, Berkeley, CA 94720-1776 (vladlen@cs.berkeley.edu).

‡School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 (michas@post.tau.ac.il). The work of this author was supported by NSF grants CCR-97-32101 and CCR-00-98246, by a grant from the U.S.–Israel Binational Science Foundation, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

$E_{\mathcal{F}}$ is itself a d-variate function whose graph is a semialgebraic set in $\mathbb{R}^{d+1}$. The projection onto d-space of this graph is called the *minimization diagram* of $\mathcal{F}$ and is denoted by $M_{\mathcal{F}}$. This is a subdivision of $\mathbb{R}^d$ into maximal connected relatively open cells of dimensions that range between 0 and d so that, for each cell τ, a fixed subset of $\mathcal{F}$ attains $E_{\mathcal{F}}$ over all points $\mathbf{x} \in \tau$ (and no other function attains the envelope over any point in τ). The complexity of $M_{\mathcal{F}}$ (and of $E_{\mathcal{F}}$) is the number of cells (of all dimensions) of $M_{\mathcal{F}}$.

It has been shown by Halperin and Sharir [12, 20] that the complexity of $M_{\mathcal{F}}$ (and of $E_{\mathcal{F}}$) is $O(n^{d+\varepsilon})$ for any $\varepsilon > 0$, where the constant of proportionality depends on ε, d, and the maximum degree of the polynomials defining the functions of $\mathcal{F}$. A (slightly) super-$\Omega(n^d)$ lower bound is known, so the bound of [12, 20] is almost tight in the worst case. Its proof is fairly involved and is based on a counting (or charging) scheme, where vertices of the envelope are charged to sets of "nearby" vertices of the arrangement $\mathcal{A}(\mathcal{F})$ of the function graphs.

This technique and various refinements and extensions thereof have been successful in bounding the complexity of lower envelopes and of several related structures, such as a single cell in a d-dimensional arrangement [6, 13]. However, the technique has had only partial success in analyzing the *overlay* of minimization diagrams. The overlay of two (or several) minimization diagrams of d-variate functions, as above, is the superposition of these diagrams in $\mathbb{R}^d$; specifically, it is the arrangement in d-space of the union of the curves or surfaces (of various dimensions) that constitute the individual diagrams. The *complexity* of the overlay is the complexity of this arrangement, namely, the number of its cells of all possible dimensions. The overlay of minimization diagrams became an important concept in the theory of arrangements after it was demonstrated that a successful analysis of the complexity of the overlay can lead to simple divide and conquer algorithms for the computation of lower envelopes and related structures [11]. Moreover, overlays arise naturally in many applications, as will be described below in more detail.

For $d = 1$ (the case of univariate functions), each minimization diagram is simply a partition of the x-axis into a finite number of intervals; if each pair of functions intersects in at most s points, then the size of the minimization diagram of n functions is known to be at most $\lambda_s(n)$ for totally defined continuous functions or $\lambda_{s+2}(n)$ for partially defined continuous functions, where $\lambda_s(n)$ is the maximum length of (n, s)-Davenport–Schinzel sequences [21], which is near-linear in n for any fixed s. The overlay of two (or more) minimization diagrams is simply the partition obtained by merging the breakpoints of the diagrams into a single sequence. The complexity of the overlay is thus proportional to the sum of the complexities of the individual diagrams; in particular, it is near-linear in the number of functions.

For $d = 2$, it was shown by Agarwal, Schwarzkopf, and Sharir [2] that the overlay of two minimization diagrams, each defined for some set of n bivariate functions of constant description complexity, is $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$. That is, the asymptotic bounds on the complexity of the overlay and on the complexity of a single diagram are the same, which is somewhat counterintuitive. The proof uses a refined and more involved variant of the counting scheme mentioned above.

The prevailing conjecture is that the complexity of the overlay of two minimization diagrams of d-variate functions of constant description complexity is $O(n^{d+\varepsilon})$ for any $\varepsilon > 0$ (the same asymptotic bound as that for the complexity of a single envelope). This has been an open problem for all $d \geq 3$. Recently [18], the authors have made a small step toward establishing the conjecture for $d = 3$, obtaining bounds of the form

$O(n^{4-\frac{1}{s}+\varepsilon})$, where s is a constant integer parameter that depends on the shape and the (constant) maximum degree of the given functions. The proof in [18] is based on the counting scheme and is highly complicated.

In this paper, we settle the conjecture affirmatively for $d = 3$ and prove the following theorem.

THEOREM 1.1. (a) *The complexity of the overlay of two minimization diagrams of a total of n trivariate functions of constant description complexity is $O(n^{3+\varepsilon})$ for any $\varepsilon > 0$.*

(b) *The complexity of the overlay of $k \geq 3$ minimization diagrams, each of n/k trivariate functions of constant description complexity, is $O(n^{3+\varepsilon})$ for any $\varepsilon > 0$.*

This is achieved by introducing the *partition technique*, a new approach to this problem, which we hope will also prove useful in the analysis of the complexity of other substructures in arrangements. The technique is based on *k-fold divide and conquer*, in which each of the given collections is partitioned into k subcollections of n/k functions each, where k is some parameter, and the complexity of the entire overlay is expressed in terms of the complexities of various "suboverlays" and related substructures. The analysis exploits and extends ideas used by Har-Peled [14, 15] for the analysis of the complexity of substructures in the overlay of planar arrangements.

We introduce our approach by presenting a new *simple* proof for the complexity of the overlay of the minimization diagrams of two collections of bivariate functions. This compares favorably with the previously available proof [2].

So far, the partition technique faces technical difficulties when applied to the analysis of other structures (like vertical decompositions) or to overlays in higher dimensions. Nevertheless, we feel hopeful that it will develop further, becoming able to tackle these problems and to find many additional applications.

Our result has several applications, which we enumerate in section 5 (some of which were already noted in our previous work [18]). It can be used to obtain an improved near-cubic bound, which is nearly tight in the worst case, on the complexity of the region enclosed between two envelopes in four dimensions. Another application is an improved near-cubic bound on the complexity of the space of all hyperplane transversals of a collection of simply shaped convex sets in 4-space, and on the complexity of the space of all line transversals of a similar collection of convex sets in 3-space. Using these bounds, one can adapt randomized incremental techniques, proposed in [1, 5], to construct the boundary of these transversal spaces in expected near-cubic time. We also obtain an improved near-cubic bound on the number of geometric permutations in a collection of disjoint convex bodies in $\mathbb{R}^3$. Our new bound can also be used to obtain a near-cubic bound on the complexity of the union of certain families of fat convex objects of nearly equal size in 4-space.

Parts of our analysis are of independent interest, and we adapt one to show that the complexity of the lower envelope of an arrangement of n totally defined semialgebraic surfaces of constant description complexity in $\mathbb{R}^3$, that does not contain any vertices, is $O(n^{1+\varepsilon})$ for any $\varepsilon > 0$.

The rest of the paper is organized as follows. We begin by introducing the partition technique. It is introduced by example, in section 2, where it is used to rederive the known near-quadratic bound for the complexity of the overlay of the minimization diagrams of two collections of bivariate functions. In section 3, we provide a useful technical tool needed for applying the partition technique in three dimensions. Finally, we use the partition technique to prove a near-cubic bound on the complexity of the overlay of minimization diagrams of trivariate functions in section 4. Several

applications of this new bound are described in section 5.

2. The overlay of bivariate minimization diagrams. We start by introducing the partition technique. This introduction is done by example, on the analysis of overlays of minimization diagrams of two collections of bivariate functions.

Let $\mathcal{F}$ and $\mathcal{G}$ be two collections, each consisting of n bivariate functions of constant description complexity. We prove that the complexity of the overlay of the minimization diagrams $M_{\mathcal{F}}$ of $\mathcal{F}$ and $M_{\mathcal{G}}$ of $\mathcal{G}$ is $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$. Denote the overlay by $Q(\mathcal{F}, \mathcal{G})$, and define the *bichromatic complexity* $C(\mathcal{F}, \mathcal{G})$ of the overlay to be the number of intersections between edges of $M_{\mathcal{F}}$ and edges of $M_{\mathcal{G}}$. Clearly, since each overlay is a planar map, the actual complexity of $Q(\mathcal{F}, \mathcal{G})$ is proportional to $C(\mathcal{F}, \mathcal{G}) + |M_{\mathcal{F}}| + |M_{\mathcal{G}}|$, where $|M_{\mathcal{F}}|$ (resp., $|M_{\mathcal{G}}|$) is the complexity of the minimization diagram $M_{\mathcal{F}}$ (resp., of $M_{\mathcal{G}}$).

Throughout what follows, we will assume that all the functions in $\mathcal{F}$ and $\mathcal{G}$ are continuous and totally defined. This involves no real loss of generality, because one can always partition each function graph in $\mathcal{F}$ and $\mathcal{G}$ into a constant number of continuous patches, which can then be extended to be totally defined without decreasing the complexity of the overlay. Indeed, a continuous partially defined function graph can be extended to be totally defined by means of near-vertical semi-infinite walls attached to its boundary. Formally, we replace each graph by the boundary of its Minkowski sum with a steeply sloped vertical cone; it is easy to see that, if the slope of the cone is large enough, this can only increase the complexity of the overlay. This extension can be analogously performed for collections $\mathcal{F}$ and $\mathcal{G}$ of functions in any dimension.

Partition $\mathcal{G}$ into k groups, $\mathcal{G}_1, \ldots, \mathcal{G}_k$, each of n/k functions, for some threshold parameter k that we will determine later. Fix an edge e of $M_{\mathcal{F}}$, and consider the vertical 2-dimensional *wall* $V^{(e)}$ erected over e; this is the union of all z-parallel lines that pass through points of e. Restrict the functions of $\mathcal{G}$ over e to obtain a collection $\mathcal{G}^{(e)}$ of univariate functions of constant description complexity. It is partitioned in an obvious way into k subcollections $\mathcal{G}_1^{(e)}, \ldots, \mathcal{G}_k^{(e)}$.

We shall now see that the complexity of the lower envelope of $\mathcal{G}^{(e)}$ is roughly proportional to the sum of the complexities of the lower envelopes of all the subsets $\mathcal{G}_i^{(e)}$, disregarding a small additive term and a near-constant multiplicative factor.

Let s denote the (constant) maximum number of intersections of the xy-projections of an intersection curve of two function graphs in $\mathcal{F}$ and of an intersection curve of two function graphs in $\mathcal{G}$. In particular, the number of intersections between any pair of (graphs of) functions in $\mathcal{G}^{(e)}$ is at most s.

Note that the lower envelope $E_{\mathcal{G}^{(e)}}$ of $\mathcal{G}^{(e)}$ is the lower envelope of the lower envelopes $E_{\mathcal{G}_i^{(e)}}$ of the subcollections $\mathcal{G}_i^{(e)}$ for $i = 1, \ldots, k$. Define the complexity $|E^{(e)}|$ of a univariate envelope $E^{(e)}$ over a connected arc e to be the number of vertices (breakpoints) of $E^{(e)}$ over points in the relative interior of e. Using an easy modification of an observation due to Har-Peled [14, 15], the complexity of $E_{\mathcal{G}^{(e)}}$ is

$$|E_{\mathcal{G}^{(e)}}| = O\left(\frac{\lambda_s(k)}{k} \sum_{i=1}^{k} \left(1 + |E_{\mathcal{G}_i^{(e)}}|\right)\right). \tag{2.1}$$

Indeed, Har-Peled's proof merges the breakpoints of the $E_{\mathcal{G}_i^{(e)}}$'s into a single sequence and partitions this sequence into blocks of k breakpoints each. Within each block, at most $2k$ functions can appear on the merged envelope, and they contribute at most $O(\lambda_s(k))$ breakpoints to $E_{\mathcal{G}^{(e)}}$. If an envelope $E_{\mathcal{G}_i^{(e)}}$ does not have any breakpoint

over (the relative interior of) e, we still need to consider the single function that it contributes to the overall envelope. The term 1, added to each term $|E_{\mathcal{G}_i^{(e)}}|$ in the above bound, takes care of these extreme cases.

Putting $\beta_s(k) = \lambda_s(k)/k$ and summing these bounds over all edges e of $M_{\mathcal{F}}$, we obtain that the bichromatic complexity $C(\mathcal{F}, \mathcal{G})$ of the overlay of $M_{\mathcal{F}}$ and $M_{\mathcal{G}}$ satisfies

$$C(\mathcal{F}, \mathcal{G}) = O\left(|M_{\mathcal{F}}|\lambda_s(k) + \beta_s(k)\sum_{i=1}^{k} C(\mathcal{F}, \mathcal{G}_i)\right). \tag{2.2}$$

Informally, we have just shown that, ignoring an additive term dominated by $|M_{\mathcal{F}}|$ and a negligible multiplicative factor of $\beta_s(k)$, the complexity of the overlay of $M_{\mathcal{F}}$ with $M_{\mathcal{G}}$ is roughly proportional to the sum of the complexities of the overlays of $M_{\mathcal{F}}$ with all the minimization diagrams $M_{\mathcal{G}_i}$. This is the essence of the partition technique for overlays, and it allows us to easily finish off the analysis as follows.

We reverse the roles of $\mathcal{F}$ and $\mathcal{G}$, as follows. Fix a subset $\mathcal{G}_i$, and consider an edge e' of $M_{\mathcal{G}_i}$. Partition $\mathcal{F}$ into k groups, $\mathcal{F}_1, \ldots, \mathcal{F}_k$, each of n/k functions, and consider the vertical 2-dimensional wall $V^{(e')}$ erected over e'. Restrict the functions of $\mathcal{F}$ over e' to obtain a collection $\mathcal{F}^{(e')}$ of univariate functions of constant description complexity, which is partitioned in an obvious way into $\mathcal{F}_1^{(e')}, \ldots, \mathcal{F}_k^{(e')}$.

As above, the lower envelope of the individual lower envelopes $E_{\mathcal{F}_j^{(e')}}$ over e' is the lower envelope $E_{\mathcal{F}^{(e')}}$. Using once again the technique of Har-Peled [14, 15], the complexity of $E_{\mathcal{F}^{(e')}}$ is

$$|E_{\mathcal{F}^{(e')}}| = O\left(\beta_s(k)\sum_{j=1}^{k}\left(1 + |E_{\mathcal{F}_j^{(e')}}|\right)\right).$$

Summing these bounds over all edges e' of $M_{\mathcal{G}_i}$, we obtain that the bichromatic complexity $C(\mathcal{F}, \mathcal{G}_i)$ of the overlay of $M_{\mathcal{F}}$ and $M_{\mathcal{G}_i}$ satisfies

$$C(\mathcal{F}, \mathcal{G}_i) = O\left(|M_{\mathcal{G}_i}|\lambda_s(k) + \beta_s(k)\sum_{j=1}^{k} C(\mathcal{F}_j, \mathcal{G}_i)\right).$$

This is essentially the same equation as (2.2), with $\mathcal{G}_i$ in the role of $\mathcal{F}$ and $\mathcal{F}$ in the role of $\mathcal{G}$, and it was obtained using an identical mechanism. If we now simply substitute this equation into (2.2), we obtain

$$C(\mathcal{F}, \mathcal{G}) = O\left(|M_{\mathcal{F}}|\lambda_s(k) + \beta_s(k)\sum_{i=1}^{k}\left(|M_{\mathcal{G}_i}|\lambda_s(k) + \beta_s(k)\sum_{j=1}^{k} C\left(\mathcal{F}_j, \mathcal{G}_i\right)\right)\right). \tag{2.3}$$

Let $C(n)$ denote the maximum complexity of the overlay of the minimization diagrams of two collections of n bivariate functions, each of the same constant description complexity,[1] and recall that the complexity of a lower envelope of n bivariate functions of constant description complexity is $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$ [12, 20]. This simplifies (2.3) into the recurrence

$$C(n) = O\left(n^{2+\varepsilon}\lambda_s(k) + k^2\beta_s^2(k)C\left(\frac{n}{k}\right)\right),$$

[1]Having a fixed constant description complexity means that a function graph is defined by a fixed maximum number of polynomials of a fixed maximum degree.

the solution of which is $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$ (see [13, 20, 21] for demonstrations of solutions of similar recurrence relations, which are obtained by choosing a suitable parameter k as a function of ε). We have thus shown the following theorem.

THEOREM 2.1. *The complexity of the overlay of the minimization diagrams of two collections of n bivariate functions of constant description complexity is $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$, where the constant of proportionality depends on ε.*

3. The partition technique in three dimensions. In this section, we take a crucial preparatory step toward a bound on the complexity of overlays of collections of trivariate functions. The partition technique, as exposed in the previous section, calls for relating the complexity of the minimization diagram of a collection $\mathcal{F} = \bigcup_{i=1}^{k} \mathcal{F}_i$ of functions to the sum of the complexities of the minimization diagrams of all the subcollections $\mathcal{F}_i$. This is precisely what was established in (2.1), in the case of univariate functions, where it was essentially shown that

$$|M_{\mathcal{F}}| = O\left(\beta_s(k) \sum_{i=1}^{k} \bigl(1 + |M_{\mathcal{F}_i}|\bigr)\right)$$

when $\mathcal{F}$ is a collection of univariate functions.

Utilizing the partition technique in three dimensions requires a parallel relation to be established for the case of bivariate functions. Such a relation, which we believe to be of independent interest, is proved below. Although it is sufficient for our purposes, it is not as intuitive as its counterpart in the univariate case.

THEOREM 3.1. *Let $\mathcal{F}_1, \ldots, \mathcal{F}_k$ be k sets of bivariate functions of constant description complexity, and put $\mathcal{F} = \bigcup_{i=1}^{k} \mathcal{F}_i$. Then*

$$|M_{\mathcal{F}}| = O\left(k^{2+\varepsilon} + k^{1+\varepsilon} \sum_{i=1}^{k} |M_{\mathcal{F}_i}| + k^{\varepsilon} \sum_{i=1}^{k} \sum_{j=i+1}^{k} C(\mathcal{F}_i, \mathcal{F}_j)\right)$$

for any $\varepsilon > 0$, where $C(\mathcal{F}_i, \mathcal{F}_j)$ is, as above, the bichromatic complexity of the overlay $Q(\mathcal{F}_i, \mathcal{F}_j)$.

Proof. $E_{\mathcal{F}}$ is the lower envelope of $E_{\mathcal{F}_1}, \ldots, E_{\mathcal{F}_k}$. Take, as above, the minimization diagrams $M_{\mathcal{F}_1}, \ldots, M_{\mathcal{F}_k}$, and overlay them to obtain a planar subdivision, which is the arrangement of the edges of these individual minimization diagrams. We may, of course, interpret this overlay as the combination of the overlays of pairs of these minimization diagrams. (For example, each vertex of the overlay, which is not a vertex of one of the diagrams $\mathcal{M}_{\mathcal{F}_i}$, is also a vertex of one of these pairwise overlays.) However, in order to derive the relation asserted in the theorem, we treat the overlay in a more "economical" manner, which can be regarded as a 2-dimensional extension of the technique of Har-Peled [14, 15].

Specifically, let N_i denote the number of edges of $M_{\mathcal{F}_i}$ for $i = 1, \ldots, k$. Clearly, $N_i = O(|M_{\mathcal{F}_i}|)$. Put $N = \sum_{i=1}^{k} N_i$, and let V denote the number of crossings between these N arcs. Note that, by definition,

$$V = \sum_{i=1}^{k} \sum_{j=i+1}^{k} C(\mathcal{F}_i, \mathcal{F}_j).$$

It is therefore sufficient to show that

$$|M_{\mathcal{F}}| = O\left(k^{2+\varepsilon} + k^{1+\varepsilon} N + k^{\varepsilon} V\right)$$

for any $\varepsilon > 0$, and we establish this bound as follows.

Put $r = \lceil N/k \rceil$, and construct a $(1/r)$-cutting Ξ of the arrangement of the above N edges. This is a decomposition of the plane into cells, each of constant description complexity, such that each cell is crossed by at most N/r arcs of the arrangement. The size of a cutting is said to be its number of cells. As shown, e.g., by de Berg and Schwarzkopf [9], there exists a cutting Ξ of size

$$O\left(r + \frac{Vr^2}{N^2}\right) = O\left(1 + \frac{N}{k} + \frac{V}{k^2}\right).$$

Let τ be a cell of Ξ. It is crossed by at most $N/r \leq k$ edges. Let m_i denote the number of edges of $M_{\mathcal{F}_i}$ that cross τ; we have $\sum_{i=1}^{k} m_i \leq k$. It is easily seen that the number of functions of $\mathcal{F}_i$ that can attain $E_{\mathcal{F}_i}$ over τ is at most $m_i + 1$. Indeed, construct a spanning tree T of the adjacency graph of the faces of $M_{\mathcal{F}_i} \cap \tau$. ($T$ exists since the adjacency graph is clearly connected.) Each edge of T corresponds to an edge of $M_{\mathcal{F}_i}$ that crosses τ, so T has at most m_i edges and thus at most $m_i + 1$ nodes, corresponding to at most $m_i + 1$ faces of $M_{\mathcal{F}_i}$ that cross τ; this is easily seen to imply the claim.

We have thus shown that the number of functions that can attain $E_{\mathcal{F}}$ over τ is at most $\sum_{i=1}^{k}(m_i + 1) \leq 2k$. The complexity of $E_{\mathcal{F}}$ over τ is thus $O(k^{2+\varepsilon})$ for any $\varepsilon > 0$ [12, 20]. Summing this bound over all cells τ of Ξ, the overall complexity of $E_{\mathcal{F}}$ is

$$O(k^{2+\varepsilon}|\Xi|) = O\left(k^{2+\varepsilon} \cdot \left(1 + \frac{N}{k} + \frac{V}{k^2}\right)\right) = O(k^{2+\varepsilon} + k^{1+\varepsilon}N + k^{\varepsilon}V)$$

for any $\varepsilon > 0$, as asserted. □

Remark 1. An obvious open problem is to extend Theorem 3.1 to the case of trivariate functions. Here we have a collection of 2-dimensional surface patches in $\mathbb{R}^3$, which are the faces of the individual minimization diagrams, and we want to construct a $(1/r)$-cutting for this collection. The crucial ingredient in the preceding proof is a sharp bound for the size of such a cutting, which becomes a considerably harder task in the trivariate case. Specifically, the only known general-purpose method for constructing cuttings of curved surfaces in three (and higher) dimensions uses the *vertical decomposition* of a sample Σ of the given surfaces (see [21]). The size of such a vertical decomposition depends on the number of *visibility events* in Σ, which are triples of the form (e, e', s), where each of e, e' is an intersection curve of two surfaces in Σ or a boundary edge or the silhouette of a single surface, and s is a vertical segment that connects a point on e to a point on e' and does not meet any other surface of Σ. Obtaining sharp bounds on the number of visibility events in a sample of faces of the k given minimization diagrams appears to be a fairly involved problem, which makes an extension of Theorem 3.1 to the trivariate case a difficult task.

4. The overlay of trivariate minimization diagrams. Armed with the extension given in Theorem 3.1, we apply the partition technique to prove the main result of the paper, which yields a near-cubic bound for the complexity of overlays of minimization diagrams of trivariate functions. The general approach is the same as the one demonstrated in the proof of Theorem 2.1, but the technical details are unfortunately more complicated.

4.1. Preliminaries. Let $\mathcal{F}$ and $\mathcal{G}$ be two collections, each consisting of n totally defined trivariate functions of constant description complexity. Consider the overlay

$Q(\mathcal{F},\mathcal{G})$ of the minimization diagrams $M_{\mathcal{F}}$ (of $\mathcal{F}$) and $M_{\mathcal{G}}$ (of $\mathcal{G}$). The combinatorial complexity of $Q(\mathcal{F},\mathcal{G})$ counts the number of cells of all dimensions in the overlay.

Each vertex (0-dimensional cell) of $Q(\mathcal{F},\mathcal{G})$ is a vertex either of $M_{\mathcal{F}}$ or of $M_{\mathcal{G}}$, or a crossing between an edge of one diagram and a 2-face of the other. Denote by $C_{32}(\mathcal{F},\mathcal{G})$ the number of crossings between edges of $M_{\mathcal{F}}$ and 2-faces of $M_{\mathcal{G}}$ (the subscripts 3 and 2 indicate that we consider interactions between features of $M_{\mathcal{F}}$ defined by three functions and features of $M_{\mathcal{G}}$ defined by two functions). Our analysis will concentrate on $C_{32}(\mathcal{F},\mathcal{G})$, and the complementary count $C_{23}(\mathcal{F},\mathcal{G})$ of the number of crossings between 2-faces of $M_{\mathcal{F}}$ and edges of $M_{\mathcal{G}}$ will be handled in a fully symmetric manner. Vertices counted in $C_{32}(\mathcal{F},\mathcal{G})$ will sometimes be referred to as $(3,2)$-vertices, and vertices in $C_{23}(\mathcal{F},\mathcal{G})$ can similarly be called $(2,3)$-vertices.

Each edge (1-dimensional cell) of $Q(\mathcal{F},\mathcal{G})$ is (a portion of) an edge either of $M_{\mathcal{F}}$ or of $M_{\mathcal{G}}$, or a maximal connected portion of an intersection curve between a 2-face of $M_{\mathcal{F}}$ and a 2-face of $M_{\mathcal{G}}$, which does not meet any other 2-face of either diagram. Any edge of the latter kind that has an endpoint can be charged to that endpoint, which is a vertex of the overlay. Under an appropriate general position assumption, each vertex is charged in this manner only $O(1)$ times, so we will not have to be concerned explicitly with such edges. Any other edge is either unbounded or a closed bounded connected curve without a vertex. An unbounded edge e, formed by the intersection of two 2-faces φ_1 of $M_{\mathcal{F}}$ and φ_2 of $M_{\mathcal{G}}$, can be charged to a crossing between φ_1 and φ_2 at infinity (or, alternatively, at some sufficiently large sphere or cube, enclosing all bounded features of $Q(\mathcal{F},\mathcal{G})$). The number of such crossings is equal to the complexity of the overlay of $M_{\mathcal{F}}$ and $M_{\mathcal{G}}$ at infinity, which is the overlay of two minimization diagrams of *bivariate* functions of constant description complexity, and its complexity, denoted as $C_\infty(\mathcal{F},\mathcal{G})$, is $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$ (see [2] and the analysis in section 2).

We are thus left with edges of Q that are closed and bounded connected components of "bichromatic" intersection curves. Computing the number of such edges turns out to be the most involved part of our analysis. We denote their number by $C_{22}(\mathcal{F},\mathcal{G})$.

There is no need to consider separately 2-faces of $Q(\mathcal{F},\mathcal{G})$. Any 2-face φ of $Q(\mathcal{F},\mathcal{G})$ can be charged to a bounding edge, except for those 2-faces that have no boundary. The number of 2-faces of this latter kind is only $O(n^2)$, as is easily seen.

Levels. We can extend the above definitions as follows. The *level* of a point $\mathbf{x}$ in the arrangement $\mathcal{A}(\mathcal{F})$ of $\mathcal{F}$ is the number of graphs of functions in $\mathcal{F}$ that lie vertically below $\mathbf{x}$, and similarly for $\mathcal{A}(\mathcal{G})$. Consider a projection of an a-level edge (resp., 2-face) of $\mathcal{A}(\mathcal{F})$, and a projection of a b-level 2-face (resp., edge) of $\mathcal{A}(\mathcal{G})$. A crossing between these two projections is said to be an (a,b)-*level overlay vertex* (note, though, that these vertices do not show up at all in the actual overlay unless $a = b = 0$). The number of such vertices is denoted by $C_{32}^{(a,b)}(\mathcal{F},\mathcal{G})$ (resp., $C_{23}^{(a,b)}(\mathcal{F},\mathcal{G})$). Denote by $C_{32}^{(\le k)}(\mathcal{F},\mathcal{G})$ (resp., $C_{23}^{(\le k)}(\mathcal{F},\mathcal{G})$) the number of all such vertices for which $a + b \le k$. $C_{22}^{(\le k)}(\mathcal{F},\mathcal{G})$ is defined analogously. (Note that, since our functions are totally defined and since we count here curves that have no vertex, the level of a curve is well defined and is the same for all points on the curve.) Obviously, $C_{32}(\mathcal{F},\mathcal{G}) = C_{32}^{(0,0)}(\mathcal{F},\mathcal{G})$, and the same is true for $C_{23}(\mathcal{F},\mathcal{G})$ and $C_{22}(\mathcal{F},\mathcal{G})$.

Denote by $C_{32}^{(\le k)}(n)$ the maximum value of $C_{32}^{(\le k)}(\mathcal{F},\mathcal{G})$ over all collections $\mathcal{F},\mathcal{G}$, each consisting of n trivariate functions of the same constant description complexity. The quantities $C_{23}^{(\le k)}(n)$, $C_{22}^{(\le k)}(n)$ are defined analogously. For the case $k = 0$, where features of the actual overlay are counted, we use the notation $C_{32}(n)$, $C_{23}(n)$, and

$C_{22}(n)$, respectively. Since $\mathcal{F}$ and $\mathcal{G}$ are assumed to belong to the same general class of functions, we have $C_{32}(n) = C_{23}(n)$ and $C_{32}^{(\leq k)}(n) = C_{23}^{(\leq k)}(n)$, so we will address only the quantities $C_{32}(n)$ and $C_{32}^{(\leq k)}(n)$, and not their symmetric counterparts, in what follows.

We can estimate $C_{32}^{(\leq k)}(n)$ in terms of $C_{32}(n)$. Since every crossing counted in $C_{32}^{(\leq k)}(\mathcal{F}, \mathcal{G})$ is defined by five surfaces, the standard random sampling argument of Clarkson and Shor [8] implies

$$C_{32}^{(\leq k)}(n) = O\left(k^5 C_{32}\left(\frac{n}{k}\right)\right). \tag{4.1}$$

Similarly, we get for overlay edges (each defined by four surfaces) that

$$C_{22}^{(\leq k)}(n) = O\left(k^4 C_{22}\left(\frac{n}{k}\right)\right). \tag{4.2}$$

Remark 2. A more general problem involving overlays of trivariate functions is that of the overlay of *three* or more minimization diagrams, rather than just two. The reason is that overlays of three minimization diagrams contain a new kind of features: vertices formed by the intersection of three 2-faces, one from each diagram. Overlays of two diagrams are special in that they do not give rise to such vertices. Nevertheless, as will be shown below, our analysis will lead us straight into the consideration of such triple overlays. As a byproduct, we will also obtain near-cubic bounds on their complexity.

4.2. Overlay vertices (counting $C_{32}(n)$). As in section 2, we apply a 2-stage analysis. We fix a parameter k, and, in the first stage, we partition $\mathcal{G}$ into k subgroups $\mathcal{G}_1, \ldots, \mathcal{G}_k$, each consisting of n/k functions.

Fix an edge e of $M_{\mathcal{F}}$, and erect a 2-dimensional wall $V^{(e)}$ over e consisting of all x_4-parallel lines that pass through e. Restrict the functions of $\mathcal{G}$ over e to obtain a collection $\mathcal{G}^{(e)}$ of univariate functions of constant description complexity, which is partitioned in an obvious way into k subcollections $\mathcal{G}_1^{(e)}, \ldots, \mathcal{G}_k^{(e)}$.

Let s denote the maximum number of intersections between the xy-projections of an intersection curve of three function graphs in $\mathcal{F}$, and of an intersection 2-surface of two function graphs in $\mathcal{G}$. Since $\mathcal{F}$ and $\mathcal{G}$ are assumed to belong to the same general class of functions, we may assume that s also bounds the number of intersections between the xy-projections of any intersection curve of three function graphs in $\mathcal{G}$, and of an intersection 2-surface of two function graphs in $\mathcal{F}$.

Consider the lower envelopes $E_{\mathcal{G}_i^{(e)}}$ of $\mathcal{G}_i^{(e)}$ for $i = 1, \ldots, k$. Note that the lower envelope of the $E_{\mathcal{G}_i^{(e)}}$'s is the restriction $E_{\mathcal{G}^{(e)}}$ of the lower envelope $E_{\mathcal{G}}$ over e and that the number of breakpoints of $E_{\mathcal{G}^{(e)}}$ is the number of crossings between e and the 2-faces of $M_{\mathcal{G}}$ (and similarly for each $E_{\mathcal{G}_i^{(e)}}$). Hence, applying the analysis of section 2, the complexity of $E_{\mathcal{G}^{(e)}}$ is

$$|E_{\mathcal{G}^{(e)}}| = O\left(\beta_s(k) \sum_{i=1}^{k} \left(1 + |E_{\mathcal{G}_i^{(e)}}|\right)\right).$$

Summing this bound over all edges e of $M_{\mathcal{F}}$, we obtain the recurrence

$$C_{32}(\mathcal{F}, \mathcal{G}) = O\left(|M_{\mathcal{F}}|\lambda_s(k) + \beta_s(k) \sum_{i=1}^{k} C_{32}(\mathcal{F}, \mathcal{G}_i)\right). \tag{4.3}$$

In the (considerably more involved) second stage, we partition $\mathcal{F}$ into k groups $\mathcal{F}_1, \ldots, \mathcal{F}_k$, each consisting of n/k functions. Fix a subset $\mathcal{G}_b$ and a 2-face φ of $M_{\mathcal{G}_b}$. Let $V^{(\varphi)}$ denote the 3-dimensional wall erected over φ. (It is the union of all x_4-parallel lines passing through points of φ.) Restrict the functions of $\mathcal{F}$ over φ to obtain a collection $\mathcal{F}^{(\varphi)}$ of bivariate functions of constant description complexity, which is partitioned in an obvious way into $\mathcal{F}_1^{(\varphi)}, \ldots, \mathcal{F}_k^{(\varphi)}$.

Note that the number of crossings between edges of $M_{\mathcal{F}}$ and φ is equal to the number of vertices of the lower envelope $E_{\mathcal{F}^{(\varphi)}}$ (over the relative interior of φ). Using Theorem 3.1, we can estimate $|E_{\mathcal{F}^{(\varphi)}}|$ or, rather, $|M_{\mathcal{F}^{(\varphi)}}|$ as follows:

$$|M_{\mathcal{F}^{(\varphi)}}| = O\left(k^{2+\varepsilon} + k^{1+\varepsilon}\sum_{i=1}^{k}|M_{\mathcal{F}_i^{(\varphi)}}| + k^{\varepsilon}\sum_{i=1}^{k}\sum_{j=i+1}^{k} C\left(\mathcal{F}_i^{(\varphi)}, \mathcal{F}_j^{(\varphi)}\right)\right).$$

We sum this bound over all 2-faces φ of $M_{\mathcal{G}_b}$ to obtain an upper bound for $C_{32}(\mathcal{F}, \mathcal{G}_b)$. The terms $k^{2+\varepsilon}$ add up to $O(k^{2+\varepsilon}|M_{\mathcal{G}_b}|)$. The sum $\sum_{\varphi}|M_{\mathcal{F}_i^{(\varphi)}}|$ for any fixed i is equal, by definition, to

$$O\left(C_{32}(\mathcal{F}_i, \mathcal{G}_b) + C_{23}(\mathcal{F}_i, \mathcal{G}_b) + C_{22}(\mathcal{F}_i, \mathcal{G}_b) + C_{\infty}(\mathcal{F}_i, \mathcal{G}_b)\right).$$

Indeed, vertices of $M_{\mathcal{F}_i^{(\varphi)}}$ that lie in the relative interior of φ are counted in $C_{32}(\mathcal{F}_i, \mathcal{G}_b)$. Closed and bounded connected edges of $M_{\mathcal{F}_i^{(\varphi)}}$ that have no vertex and are fully contained in the relative interior of φ are counted in $C_{22}(\mathcal{F}_i, \mathcal{G}_b)$. Edges with no vertex that reach the boundary of φ induce, at their boundary crossings, $(2,3)$-vertices and are thus counted in $C_{23}(\mathcal{F}_i, \mathcal{G}_b)$. Edges that reach infinity (which can happen when φ is unbounded) are counted in $C_{\infty}(\mathcal{F}_i, \mathcal{G}_b)$. Finally, edges with a vertex can be charged to that vertex.

The sum $\sum_{\varphi} C(\mathcal{F}_i^{(\varphi)}, \mathcal{F}_j^{(\varphi)})$ for any fixed i, j is equal to the number of vertices of the *triple overlay* of $M_{\mathcal{F}_i}$, $M_{\mathcal{F}_j}$, and $M_{\mathcal{G}_b}$, which are intersections of three 2-faces, one of each diagram; we refer to such vertices as *trichromatic*. We denote this triple overlay by $Q^*(\mathcal{F}_i, \mathcal{F}_j, \mathcal{G}_b)$ and denote by $C_{222}(\mathcal{F}_i, \mathcal{F}_j, \mathcal{G}_b)$ the number of trichromatic vertices of the overlay. We define the notation $C_{222}^{(\le k)}(\mathcal{F}_i, \mathcal{F}_j, \mathcal{G}_b)$, $C_{222}^{(\le k)}(n)$, and $C_{222}(n)$ in complete analogy with the definition of the similar quantities given above.

Note that we started our analysis by considering the overlay $Q(\mathcal{F}, \mathcal{G})$ of only two minimization diagrams, and there we needed only to consider "bichromatic" vertices, formed by the intersection of edges of one diagram with the 2-faces of the other. As mentioned in the remark above, when we overlay three or more diagrams, we also encounter trichromatic vertices, formed by the intersection of three 2-faces, one of each diagram.

Combining the analysis just given with (4.3) and using the fact that $|M_{\mathcal{F}}| = O(n^{3+\varepsilon})$ for any $\varepsilon > 0$ [20], we obtain, for any $\varepsilon > 0$,

$$\begin{aligned} C_{32}(n) &= O\left(\lambda_s(k)n^{3+\varepsilon} + \beta_s(k)\sum_{b=1}^{k}\left[k^{2+\varepsilon}|M_{\mathcal{G}_b}|\right.\right. \\ &\qquad \left.\left. + k^{1+\varepsilon}\sum_{i=1}^{k}\left(C_{32}\left(\frac{n}{k}\right) + C_{22}\left(\frac{n}{k}\right) + \left(\frac{n}{k}\right)^{2+\varepsilon}\right) + k^{\varepsilon}\sum_{i=1}^{k}\sum_{j=i+1}^{k} C_{222}\left(\frac{n}{k}\right)\right]\right) \\ &= O\left(\lambda_s(k)n^{3+\varepsilon} + k^{3+\varepsilon}C_{32}\left(\frac{n}{k}\right) + k^{3+\varepsilon}C_{22}\left(\frac{n}{k}\right) + k^{3+\varepsilon}C_{222}\left(\frac{n}{k}\right)\right). \end{aligned} \tag{4.4}$$

Note that the final value of ε in this relation has to be taken slightly larger than the one we started with to accommodate the extra factor $\beta_s(n)$. However, the recurrence still holds for any arbitrarily small $\varepsilon > 0$.

4.3. Trichromatic vertices (counting $C_{222}(n)$). Let $\mathcal{F}, \mathcal{G}, \mathcal{H}$ be three collections, each consisting of n trivariate functions of constant description complexity, as above. We want to estimate the number $C_{222}(\mathcal{F}, \mathcal{G}, \mathcal{H})$ of triple intersections of 2-faces in the triple overlay $Q^*(\mathcal{F}, \mathcal{G}, \mathcal{H})$ of $M_{\mathcal{F}}, M_{\mathcal{G}}, M_{\mathcal{H}}$. Here we use a 3-stage analysis based on a variant of the analysis of the overlay of bivariate functions given in section 2. Interestingly, this part of our analysis of overlays of trivariate minimization diagrams is the simplest. Informally, this is because, in each of the three stages below, we have to consider only overlays of univariate functions, as in section 2.

We fix a parameter k and partition $\mathcal{H}$ into k groups $\mathcal{H}_1, \ldots, \mathcal{H}_k$ of n/k functions each. Fix a 2-face φ_1 of $M_{\mathcal{F}}$ and a 2-face φ_2 of $M_{\mathcal{G}}$, and let e be a connected component of the intersection curve $\varphi_1 \cap \varphi_2$ in $\mathbb{R}^3$. Note that the number of such edges e that have at least one endpoint is proportional to $C_{32}(\mathcal{F}, \mathcal{G}) + C_{23}(\mathcal{F}, \mathcal{G})$, because each endpoint of e is a crossing between an edge of one diagram and a 2-face of the other. Any other edge is either unbounded or a closed and bounded Jordan curve, and the number of such edges, as analyzed above, is $O(n^{2+\varepsilon} + C_{22}(n))$. Thus the number of edges e under consideration is $O(n^{2+\varepsilon} + C_{32}(n) + C_{22}(n))$.

Restrict the functions in $\mathcal{H}$ to the vertical wall $V^{(e)}$, as defined above, to obtain a collection $\mathcal{H}^{(e)}$ of univariate functions of constant description complexity, which is partitioned in an obvious way into k subcollections $\mathcal{H}_1^{(e)}, \ldots, \mathcal{H}_k^{(e)}$. As above, we have

$$|M_{\mathcal{H}^{(e)}}| = O\left(\beta_s(k) \sum_{i=1}^{k} \left(1 + |M_{\mathcal{H}_i^{(e)}}|\right)\right),$$

and, summing this bound over all such edges e, we obtain, by definition,

$$C_{222}(\mathcal{F}, \mathcal{G}, \mathcal{H}) = O\left(\left(n^{2+\varepsilon} + C_{22}(n) + C_{32}(n)\right) \lambda_s(k) + \beta_s(k) \sum_{i=1}^{k} C_{222}(\mathcal{F}, \mathcal{G}, \mathcal{H}_i)\right).$$

We repeat the same counting stage twice more. In the second stage, we fix $\mathcal{F}$ and one subcollection $\mathcal{H}_c$, partition $\mathcal{G}$ into k subgroups $\mathcal{G}_1, \ldots, \mathcal{G}_k$, and conclude, as above, that

$$C_{222}(\mathcal{F}, \mathcal{G}, \mathcal{H}_c) = O\left(\left(n^{2+\varepsilon} + C_{22}(n) + C_{32}(n)\right) \lambda_s(k) + \beta_s(k) \sum_{j=1}^{k} C_{222}(\mathcal{F}, \mathcal{G}_j, \mathcal{H}_c)\right).$$

(Note that the "overhead" term depends on $C_{22}(\mathcal{F}, \mathcal{H}_c) + C_{32}(\mathcal{F}, \mathcal{H}_c) + C_{23}(\mathcal{F}, \mathcal{H}_c) + C_{\infty}(\mathcal{F}, \mathcal{H}_c)$, which still involves $\Theta(n)$ functions in $\mathcal{F}$.)

Similarly, in the third stage, we fix two subsets $\mathcal{G}_b$ and $\mathcal{H}_c$, partition $\mathcal{F}$ into k subgroups $\mathcal{F}_1, \ldots, \mathcal{F}_k$, and obtain

$$C_{222}(\mathcal{F}, \mathcal{G}_b, \mathcal{H}_c) = O\left(\left(\left(\frac{n}{k}\right)^{2+\varepsilon} + C_{22}\left(\frac{n}{k}\right) + C_{32}\left(\frac{n}{k}\right)\right) \lambda_s(k) + \beta_s(k) \sum_{\ell=1}^{k} C_{222}\left(\mathcal{F}_\ell, \mathcal{G}_b, \mathcal{H}_c\right)\right).$$

Substituting these estimates into one another, we obtain

$$\begin{aligned} C_{222}(n) = O\Bigg(& \left(n^{2+\varepsilon} + C_{22}(n) + C_{32}(n)\right)\lambda_s(k) \\ & + \lambda_s(k)\Bigg[\left(n^{2+\varepsilon} + C_{22}(n) + C_{32}(n)\right)\lambda_s(k) \\ & + \lambda_s(k)\left[\left(\left(\frac{n}{k}\right)^{2+\varepsilon} + C_{22}\left(\frac{n}{k}\right) + C_{32}\left(\frac{n}{k}\right)\right)\lambda_s(k) + \lambda_s(k) C_{222}\left(\frac{n}{k}\right)\right]\Bigg]\Bigg) \end{aligned}$$

or

$$\begin{aligned} & C_{222}(n) \\ & = O\left(\lambda_s^2(k)\left[n^{2+\varepsilon} + C_{22}(n) + C_{32}(n)\right] + \lambda_s^3(k)\left[C_{22}\left(\frac{n}{k}\right) + C_{32}\left(\frac{n}{k}\right) + C_{222}\left(\frac{n}{k}\right)\right]\right). \end{aligned} \tag{4.5}$$

4.4. Overlay edges (counting $C_{22}(n)$). In this section, we analyze the quantity $C_{22}(\mathcal{F}, \mathcal{G})$, which is the number of "bichromatic" overlay edges that are closed bounded Jordan curves, each formed as a connected component of an intersection of a face of $M_{\mathcal{F}}$ and a face of $M_{\mathcal{G}}$, which are not adjacent to any overlay vertex. Overlay edges are structurally quite different from overlay vertices. To bound their number, we do not use the partition technique but employ the well-known technique of counting schemes introduced by Halperin and Sharir [12, 20] (see also [21]) and already mentioned in the introduction. This part of the analysis is essentially identical to the corresponding part in the precursor paper [18].

Our counting scheme for $C_{22}(\mathcal{F}, \mathcal{G})$ is another novel technical feature of this paper, and no similar scheme that charges curves rather than vertices has been employed in previous works involving substructures in arrangements. In this counting scheme, we partition the set of overlay edges into a small number of groups, according to the *index* of the edge (a notion to be defined shortly, and quite different from the standard notion of indices, as used in the analysis of substructures in arrangements [21]). We establish a separate recurrence relation for the number of edges in each group. These recurrences will depend on each other as well as on $C_{32}(n)$.

Fix some threshold parameter k. Consider the (at most a constant number of) connected components of a bichromatic overlay intersection curve defined by a fixed quadruple of surfaces $f_1, f_2 \in \mathcal{F}$, $g_1, g_2 \in \mathcal{G}$. Suppose one of these components contains a point at level at most k and either is incident to an $(\leq k)$-level overlay vertex or extends to infinity. The discussion in section 4.1 easily implies that the number of such overlay edges (that is, edges counted in $C_{22}(n)$ for which a sibling component of the same intersection curve satisfies the above properties) is, for any $\varepsilon > 0$,

$$O\left(k^5 C_{32}\left(\frac{n}{k}\right) + k^2 n^{2+\varepsilon}\right). \tag{4.6}$$

In the remainder of this section, we treat bichromatic overlay edges defined by some quadruple f_1, f_2, g_1, g_2 such that all $(\leq k)$-level edges defined by f_1, f_2, g_1, g_2 are closed and bounded Jordan curves that are not incident to any vertex. (The level of such a curve is well defined: all points on the curve have the same level, assuming, as above, that the functions in $\mathcal{F}$ and $\mathcal{G}$ are totally defined.) We fix a 2-face φ that belongs to $E_{\mathcal{G}}$ and is a connected portion of the intersection of the graphs of two functions $g_1, g_2 \in \mathcal{G}$, and we consider the vertical 3-dimensional wall $V^{(\varphi)}$ erected over φ, as in the previous sections. Let $\mathcal{F}^{(\varphi)}$ be the cross-section of $\mathcal{F}$ within $V^{(\varphi)}$,

and let f_φ be the cross-section of the graph of a function f within $V^{(\varphi)}$ for each $f \in \mathcal{F}$. $\mathcal{A}(\mathcal{F}^{(\varphi)})$ can be regarded as an arrangement of xy-monotone 2-dimensional surfaces in $\mathbb{R}^3$.

Let $\Gamma(\mathcal{F}^{(\varphi)})$ denote the set of edges, for which the following holds. Each edge c of $\Gamma(\mathcal{F}^{(\varphi)})$ is a 0-level edge that lies completely in the lower envelope of $\mathcal{A}(\mathcal{F}^{(\varphi)})$ and is a connected component of an intersection curve $f_1 \cap f_2$ for some $f_1, f_2 \in \mathcal{F}^{(\varphi)}$ such that all the connected components (including c) of this intersection that lie at level at most k of $\mathcal{A}(\mathcal{F}^{(\varphi)})$ are closed and bounded Jordan curves and are not incident to any vertex. Note that any bichromatic overlay edge of the kind we are after corresponds to a curve in $\Gamma(\mathcal{F}^{(\varphi)})$ for some 2-face φ of $E_\mathcal{G}$.

Define $S(\mathcal{F}, \mathcal{G}) \equiv \sum_\varphi |\Gamma(\mathcal{F}^{(\varphi)})|$ and $S(n) \equiv \max S(\mathcal{F}, \mathcal{G})$, where the maximum is taken over all sets $\mathcal{F}$ and $\mathcal{G}$ of size n as above. The above analysis implies, for any $\varepsilon > 0$,

$$C_{22}(n) = O\left(S(n) + k^5 C_{32}\left(\frac{n}{k}\right) + k^2 n^{2+\varepsilon}\right). \tag{4.7}$$

We next define the notion of *index* that we attach to intersection curves. For readers who have encountered previous similar-purpose definitions [20], we remark that our definition is conceptually different in that it is not "local," meaning that the index given to a specific intersection curve is defined with respect to the whole arrangement $\mathcal{A}(\mathcal{F}^{(\varphi)})$ and may change as the set $\mathcal{F}^{(\varphi)}$ decreases.

Specifically, consider the intersection $f_1 \cap f_2$ for any $f_1, f_2 \in \mathcal{F}^{(\varphi)}$. Consider all the connected components of this intersection that are edges of $\Gamma(\mathcal{F}^{(\varphi)})$, and let j be their number. We set the index of the curve $f_1 \cap f_2$ to j. We will say that j is also the index of all the edges of $\Gamma(\mathcal{F}^{(\varphi)})$ that are connected components of $f_1 \cap f_2$.

Let q be the maximum possible number of connected components of an intersection $f_1 \cap f_2$ as above. By the constant description complexity and the general position assumptions, q is constant. Clearly, j varies between 1 and q. The case $j = 0$ means that either no component of $f_1 \cap f_2$ shows up on the envelope or some such components do show up, but there exists a component at level at most k that has a vertex or reaches infinity. We will not be concerned with edges of index 0, because either we do not have to count them at all, or else we can bound their number using (4.6).

Let $\Gamma^{(j)}(\mathcal{F}^{(\varphi)})$ denote the subset of $\Gamma(\mathcal{F}^{(\varphi)})$ that contains edges with index at least j. Define $S^{(j)}(\mathcal{F}, \mathcal{G}) \equiv \sum_\varphi |\Gamma^{(j)}(\mathcal{F}^{(\varphi)})|$ and $S^{(j)}(n) \equiv \max S^{(j)}(\mathcal{F}, \mathcal{G})$, where the maximum is taken over all sets $\mathcal{F}$ and $\mathcal{G}$ of size n as above. Since the maximal index of an edge is q, we have $S^{(q+1)}(n) = 0$. We also have, by definition, $S(n) = S^{(1)}(n)$.

We note that the index of $f_1 \cap f_2$ depends on the current set $\mathcal{F}$. When $\mathcal{F}$ is replaced by a smaller sample, as happens, for example, when applying the Clarkson–Shor bound, the index may increase, because either (i) more components of $f_1 \cap f_2$ appear on the envelope, or (ii) all vertices of $\mathcal{A}(\mathcal{F}^{(\varphi)})$ that lie on components of $f_1 \cap f_2$ at level at most k disappear, since all the third surfaces incident to these vertices have been removed from $\mathcal{F}$. (Note that the level of a curve can only decrease when functions are removed from $\mathcal{F}$.) In this latter case, the index jumps from 0 (no component of $f_1 \cap f_2$, even those on the envelope, qualified to belong to $\Gamma(\mathcal{F}^{(\varphi)})$ before $\mathcal{F}$ was reduced) to some j equal to the number of components that lie on the envelope after the reduction.

The index of $f_1 \cap f_2$ can decrease in only one way, as follows. There may exist an intersection curve $f_1 \cap f_2$ that had a positive index, but, after $\mathcal{F}$ has been reduced, new components of $f_1 \cap f_2$ reach the $(\leq k)$-level of $\mathcal{A}(\mathcal{F}^{(\varphi)})$ and do contain vertices or do reach infinity. The index of $f_1 \cap f_2$ then drops to 0. However, the number of

such curves in the reduced arrangement can be estimated using the bound in (4.6) (applied to the reduced arrangement). To conclude, with the exception of these drops to zero, the index of a curve can only increase when the size of $\mathcal{F}$ is reduced.

The counting scheme below bounds $S^{(j)}(n)$ for all $1 \leq j \leq q$. It proceeds by distinguishing between five possible scenarios (Cases 1–5) and treating each in turn.

Case 1: $\Gamma^{(j)}(\mathcal{F}^{(\varphi)})$ *is small.* Suppose first that at most $(q+1)k+2 = O(k)$ surfaces of $\mathcal{F}$ attain the lower envelope of $\mathcal{A}(\mathcal{F}^{(\varphi)})$. In this case, we use the naive bound $|\Gamma^{(j)}(\mathcal{F}^{(\varphi)})| = O(k^2)$. Since there are $O(n^{3+\varepsilon})$ possible faces φ, the maximum number of edges of this kind that are counted in $S^{(j)}(n)$ is $O(k^2 n^{3+\varepsilon})$ (for all j). In what follows, we consider only faces φ such that more than $(q+1)k+2$ surfaces of $\mathcal{F}$ attain the lower envelope of $\mathcal{A}(\mathcal{F}^{(\varphi)})$.

Case 2: *There is a "shallow" connected component of the same intersection curve.* Consider any pair of surfaces $P, Q \in \mathcal{F}$ such that there is a connected component c of the intersection $P_\varphi \cap Q_\varphi$ that is an edge of $\Gamma^{(j)}(\mathcal{F}^{(\varphi)})$. The component c is, by definition, also an edge of $\Gamma(\mathcal{F}^{(\varphi)})$, which implies that all the connected components of $P_\varphi \cap Q_\varphi$ that lie at level at most k of $\mathcal{A}(\mathcal{F}^{(\varphi)})$ are closed and bounded Jordan curves and are not incident to a vertex. Also, since the index of $P_\varphi \cap Q_\varphi$ is at least j, the number of connected components of $P_\varphi \cap Q_\varphi$ that lie on the lower envelope of $\mathcal{A}(\mathcal{F}^{(\varphi)})$ is at least j. Suppose there is a connected component c' defined by $P_\varphi \cap Q_\varphi$ whose level is between 1 and k. (As noted, the level of a curve that satisfies the above properties is well defined.) In this case, we charge all edges of $\Gamma^{(j)}(\mathcal{F}^{(\varphi)})$ defined by $P_\varphi \cap Q_\varphi$ to c'. It is easy to see that each such edge c' is charged in this fashion at most a constant number of times (that is, at most $q-1$ times).

Let $\Gamma(c')$ be the set of the (at most k) surfaces of $\mathcal{F}^{(\varphi)}$ that lie below c'. Set $\tilde{\mathcal{F}}^{(\varphi)} \equiv \mathcal{F}^{(\varphi)} \setminus \Gamma(c')$, and consider the arrangement $\mathcal{A}(\tilde{\mathcal{F}}^{(\varphi)})$. Clearly, c' lies on its lower envelope. Moreover, there are at least $j+1$ connected components of the intersection $P_\varphi \cap Q_\varphi$ that lie on this lower envelope. Thus the index of c' is now at least $j+1$, and c' belongs to $\Gamma^{(j+1)}(\tilde{\mathcal{F}}^{(\varphi)})$. More accurately, c' belongs to $\Gamma^{(j+1)}(\tilde{\mathcal{F}}^{(\varphi)})$ unless a new component of $P_\varphi \cap Q_\varphi$, that either reaches infinity or contains a vertex of $\mathcal{A}(\tilde{\mathcal{F}}^{(\varphi)})$, has "descended" to the first k levels in $\mathcal{A}(\tilde{\mathcal{F}}^{(\varphi)})$, thereby dropping the index of c' to 0. (Note that this analysis also applies to any subset $\mathcal{F}' \subseteq \tilde{\mathcal{F}}^{(\varphi)}$.) A standard random sampling argument, such as the ones used in section 4.1, now implies, in combination with (4.6), that the maximum number of edges of this kind that are counted in $S^{(j)}(\mathcal{F}, \mathcal{G})$ is

$$O\left(k^4 S^{(j+1)}\left(\frac{n}{k}\right) + k^4\left[k^5 C_{32}\left(\frac{n}{k^2}\right) + n^{2+\varepsilon}\right]\right) = O\left(k^4 S^{(j+1)}\left(\frac{n}{k}\right) + k^9 C_{32}\left(\frac{n}{k^2}\right) + k^4 n^{2+\varepsilon}\right).$$

In what follows, we assume that there is no connected component c' as above.

Case 3: *There is a "shallow" vertex.* Since Case 1 is ruled out, there are at least $(q+1)k$ surfaces, other than P_φ, Q_φ, that attain the lower envelope of $\mathcal{A}(\mathcal{F}^{(\varphi)})$ over φ. Consider all the connected components (edges) of $P_\varphi \cap Q_\varphi$ that lie in the lower envelope of $\mathcal{A}(\mathcal{F}^{(\varphi)})$. These edges partition both P_φ and Q_φ into at most $(q+1)$ pairs of relatively open regions, where each pair consists of two regions, one on P_φ and one on Q_φ, that have a common boundary (over the relative interior of φ) and a common projection onto φ. One of these projections, denoted by Δ_0, has to contain at least k subregions where k other distinct surfaces attain the envelope. Each of these surfaces, T_φ, lies strictly above $P_\varphi \cap Q_\varphi$ over all points of $\partial\Delta_0$ defined by $P_\varphi \cap Q_\varphi$. Hence T_φ intersects both P_φ and Q_φ over Δ_0, and the projection of each component

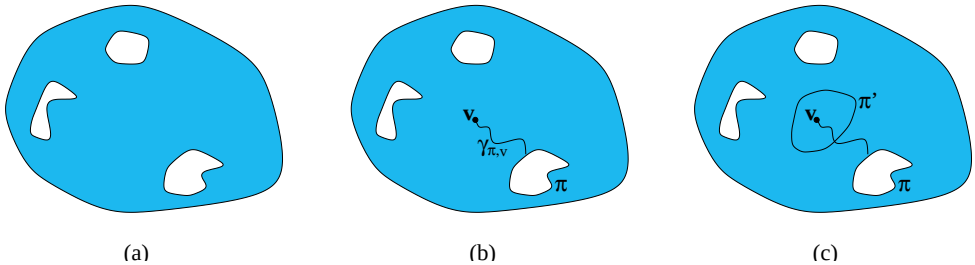


FIG. 4.1. *The region* Δ *(shaded) is illustrated in* (a). (b) *shows a point* v *that is charged by an edge* π*, together with the connecting arc* $\gamma_{\pi,v}$*. The interior of the arc* $\gamma_{\pi,v}$ *lies fully within* Δ *and thus cannot intersect any connected component of* $P_\varphi \cap Q_\varphi$ *that lies on the lower envelope.* (c) *depicts the edge* π'*, whose assumed existence leads to the contradiction in the proof of Lemma* 4.1.

of either intersection onto φ is not incident to those boundary components of Δ_0 that are defined by $P_\varphi \cap Q_\varphi$. (We emphasize that the region Δ_0 is *fixed* in all the remaining charging steps during the present case and in Cases 4 and 5 below.)

Consider an arbitrary edge π (defined by $P_\varphi \cap Q_\varphi$) whose projection onto φ lies on the boundary of Δ_0, and assume, without loss of generality, that P_φ lies below Q_φ when approaching (the projection of) π from within Δ_0. Let Δ denote the portion of Q_φ (the surface that is higher near π) that projects onto Δ_0 (see Figure 4.1(a)), and let C be the set of edges (each being a connected component of $O_\varphi \cap Q_\varphi$ for some $O_\varphi \in \mathcal{F}^{(\varphi)}$) contained in Δ. By what we have just argued, $|C| \geq k$.

Suppose that there is at least one vertex v on Q_φ within Δ that lies at level at most k, such that v can be connected to a point on π by an arc $\gamma_{\pi,v}$ (of some finite though not necessarily constant description complexity) that lies on Q_φ within Δ, and is at level ($\leq k$) for all points in its relative interior. In this case, we charge all the edges of $\Gamma^{(j)}(\mathcal{F}^{(\varphi)})$ defined by $P_\varphi \cap Q_\varphi$ to an arbitrary such vertex (see Figure 4.1(b)). Note that each such vertex corresponds to a $(\leq k, 0)$-level overlay vertex.

Similarly, suppose there is at least one edge that passes above/below the boundary of φ at level $\leq k$, while lying on Q_φ within Δ, at a certain point v, such that v can be connected to π by an arc $\gamma_{\pi,v}$ as above. By construction, this implies that part of $\partial\Delta$ is covertical with the boundary of φ, and this is the part of $\partial\Delta$ that contains v. (In general, notice that Δ is always bounded by φ in the sense that the projection Δ_0 of Δ is contained in φ. In the case under consideration, Δ_0 touches the boundary of φ, and thus part of $\partial\Delta_0$ lies on $\partial\varphi$.) In this case, we charge all the edges in $\Gamma^{(j)}(\mathcal{F}^{(\varphi)})$ defined by $P_\varphi \cap Q_\varphi$ to an arbitrary such point v, as above, noting that each such point again corresponds to a $(\leq k, 0)$-level overlay vertex. (Note that, in the preceding case, the charged vertex was a $(3, 2)$-vertex, whereas now it is a $(2, 3)$-vertex.)

An observation that will prove useful in the proof of the following lemma is that an arc $\gamma_{\pi,v}$ cannot cross any connected component of $P_\varphi \cap Q_\varphi$ that lies in the lower envelope of $\mathcal{A}(\mathcal{F}^{(\varphi)})$, since the interior of $\gamma_{\pi,v}$ is required to lie inside the region Δ.

LEMMA 4.1. *Each point* v *is charged by at most* qk *distinct edges of* $\Gamma^{(j)}(\mathcal{F}^{(\varphi)})$ *as prescribed in Case* 3.

Proof. The proof is visualized in Figure 4.1. Suppose a point v that lies on Q_φ is charged by an edge π. By construction, π is a connected component of an intersection of Q_φ with another surface P_φ. The crucial observation is that P_φ has to lie below v. Indeed, suppose P_φ lies above v. By construction, there exists an arc $\gamma_{\pi,v}$ that connects v to a point on π and lies fully within level $\leq k$ in its interior. Also by

construction, P_φ lies below Q_φ when we approach π on $\gamma_{\pi,v}$, and our assumption states that P_φ lies above Q_φ when we approach v on $\gamma_{\pi,v}$. Thus P_φ has to intersect the interior of $\gamma_{\pi,v}$, which implies the existence of an edge π' defined by $P_\varphi \cap Q_\varphi$, distinct from π, within level $\leq k$. As observed just before the statement of the lemma, $\gamma_{\pi,v}$ cannot intersect π' at level 0. Thus the level of π' is between 1 and k. Such a situation, however, has been ruled out in Case 2, leading us to a contradiction.

We have thus shown that v can only be charged by connected components of intersections of Q_φ with surfaces that lie below v. Since at most k surfaces lie below v and each defines at most q such connected components (edges) along Q_φ, v is charged by at most qk distinct edges that lie on Q_φ. Since v lies on at most three surfaces of $\mathcal{F}^{(\varphi)}$, it can be charged by at most $3qk$ edges overall. □

Combined with (4.1) and with the fact that q is a constant, Lemma 4.1 implies that the maximum number of edges of this kind that are counted in $S^{(j)}(\mathcal{F}, \mathcal{G})$ is $O(kC_{32}^{(\leq k)}(n)) = O(k^6 C_{32}(n/k))$. In what follows, we assume that there is no vertex v as above (for the specific region Δ which we now keep fixed).

Case 4: *There is a "shallow" edge that reaches infinity.* We continue to use the setup introduced in Case 3 and suppose that there is an edge c of C that lies at level at most k and is not a closed and bounded Jordan curve, and c can be reached from π along an arc $\gamma_{\pi,c}$, as above, that stays at level $\leq k$. (Since we assume that the scenario treated in Case 3 does not hold, this can only occur if Δ is unbounded, c has no vertices, and c reaches infinity. This, in turn, can only happen when φ is unbounded.) In this case, we charge all the edges of $\Gamma^{(j)}(\mathcal{F}^{(\varphi)})$ defined by $P_\varphi \cap Q_\varphi$ to the edge c. Arguing as above, we can show that the overall number of such edges is $O(k^2 n^{2+\varepsilon})$. The proof of Lemma 4.1 can easily be modified to show that each edge is charged at most $2qk$ times in this fashion. (In the modified proof, we use the fact that the set of surfaces that lie below a point on c is the same for all points of c.) Thus the maximum number of edges of this kind that are counted in $S^{(j)}(\mathcal{F}, \mathcal{G})$ is $O(k^3 n^{2+\varepsilon})$. In what follows, we assume that there is no edge $c \in C$ as above (for our fixed Δ).

Case 5. In this final case, we distinguish between two subcases. In both, we charge one edge π of $\Gamma^{(j)}(\mathcal{F}^{(\varphi)})$ defined by $P_\varphi \cap Q_\varphi$ (out of the at most q such edges), which bounds the region Δ under consideration, to k edges $c \in C$ that lie at level $\leq k$ and can be reached from π along an $(\leq k)$-level arc $\gamma_{\pi,c}$, as above.

Subcase 5.A: *There is a point on* Δ *that lies at level* $> k$. We can connect this point to a point on one of the boundary arcs π of Δ, defined by $P_\varphi \cap Q_\varphi$, by an arc (of some finite though not necessarily constant description complexity) that lies on Q_φ within Δ, in its interior. Since one endpoint of this arc lies at level $> k$ and the other lies on the lower envelope, the arc intersects at least k distinct edges of C. Moreover, the first k distinct edges encountered when walking along the arc away from π lie at level $\leq k$ since π lies on the lower envelope. We charge π to these first k edges of C.

Subcase 5.B: Δ *lies entirely at level* $\leq k$. We charge π to k arbitrary edges of C.

We emphasize that in both subcases there exists an arc $\gamma_{\pi,c}$ for each edge c charged by π that connects a point on c to a point on π and lies on Q_φ within Δ, and its interior lies fully at level $\leq k$. In Subcase 5.A, $\gamma_{\pi,c}$ is the appropriate prefix of the arc used to identify the k edges that are charged, while in Subcase 5.B, $\gamma_{\pi,c}$ exists since Δ lies entirely at level $\leq k$ and is connected. Therefore, since Cases 1–4 are assumed not to occur, it follows that each of these edges c is a closed bounded Jordan curve not incident to any vertex. Indeed, had vertices incident to c existed, one of them would lie at level $\leq k$ and could be connected to a point on π as prescribed in Case 3. We have assumed, however, that there are no such vertices. The exclusion of Case

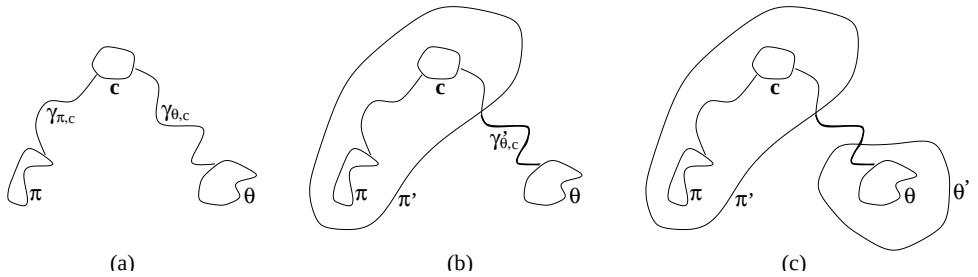


FIG. 4.2. *A schematic visualization of the proof of Lemma* 4.2. *The general setup is introduced in* (a)*;* (b) *illustrates the edge* π' *and the arc* $\gamma'_{\theta,c}$ *(with the latter thickened), and* (c) *illustrates the edge* θ'.

4 similarly implies that c is a closed bounded Jordan curve. (Note that there may nevertheless exist edges $c \in C$ at level $\leq k$ that contain a vertex or are unbounded but are *unreachable* from π along a low-level arc $\gamma_{\pi,c}$, as above.)

Let c be an edge of C that is charged in the above fashion. Lemma 4.2 below states that c is charged at most twice. Let $\Gamma(c)$ be the set of the (at most k) surfaces of $\mathcal{F}^{(\varphi)}$ that lie below c, and set $\tilde{\mathcal{F}}^{(\varphi)} \equiv \mathcal{F}^{(\varphi)} \setminus \Gamma(c)$. Since we assume that none of the scenarios treated in Cases 1–4 holds, it is easy to see that c is an edge of $\Gamma(\tilde{\mathcal{F}}^{(\varphi)})$, unless there exists a component c' "sibling" to c that meets the first k levels of $\mathcal{A}(\tilde{\mathcal{F}}^{(\varphi)})$ and either reaches infinity or contains a vertex of $\mathcal{A}(\tilde{\mathcal{F}}^{(\varphi)})$; the component c' has either descended to the first k levels in $\mathcal{A}(\tilde{\mathcal{F}}^{(\varphi)})$, thereby dropping the index of c to 0, or has already existed within the first k levels of $\mathcal{A}(\mathcal{F}^{(\varphi)})$ but was "hidden" from the charging curve π, as described in the preceding paragraph. (These properties also hold for any subset of $\tilde{\mathcal{F}}^{(\varphi)}$.) As in Case 2, a standard random sampling argument, such as the ones used in section 4.1, now implies, in combination with (4.6), that the maximum number of edges of this kind that are counted in $S^{(j)}(\mathcal{F}, \mathcal{G})$ is

$$\frac{q}{k} \cdot O\left(k^4 S\left(\frac{n}{k}\right) + k^4\left[k^5 C_{32}\left(\frac{n}{k^2}\right) + n^{2+\varepsilon}\right]\right) = O\left(k^3 S\left(\frac{n}{k}\right) + k^8 C_{32}\left(\frac{n}{k^2}\right) + k^3 n^{2+\varepsilon}\right).$$

We now give the lemma, referred to in the beginning of the paragraph, that ensures that each edge $c \in C$ is charged in the above fashion at most twice.

LEMMA 4.2. *Each curve is charged by at most two distinct edges of* $\Gamma^{(j)}(\mathcal{F}^{(\varphi)})$ *as prescribed in Case* 5.

Proof. By construction, an edge $c \subseteq Q_\varphi \cap O_\varphi$ can only be charged by edges that lie either on Q_φ or on O_φ. Assume, for the sake of contradiction, that c is charged by two edges, $\pi \subseteq Q_\varphi \cap P_\varphi$ and $\theta \subseteq Q_\varphi \cap T_\varphi$, for some $P_\varphi, T_\varphi \in \mathcal{F}^{(\varphi)}$ (see Figure 4.2). Notice that the level of c is at most k and that c can be connected to π and θ by arcs $\gamma_{\pi,c}$ and $\gamma_{\theta,c}$, respectively, as described above (see Figure 4.2(a) for an illustration). As argued above, this implies that c is a closed bounded Jordan curve that is incident to no vertex.

We can assume, without loss of generality, that θ and c lie on the same side of the closed curve π. The arguments in the proof of Lemma 4.1 imply that P_φ has to lie completely below c. Consider the arc $\gamma_{\theta,c}$ connecting a point on c to a point on θ, as above. P_φ lies below Q_φ when we approach c on $\gamma_{\theta,c}$, but since θ lies on the lower envelope, P_φ lies above Q_φ when we approach θ on $\gamma_{\theta,c}$. Thus P_φ has to intersect the

relative interior of $\gamma_{\theta,c}$, which implies the existence of a closed curve π' defined by $P_\varphi \cap Q_\varphi$, within level $\leq k$ such that θ and c lie on different sides of π'. π' is therefore distinct from π. If its level is between 1 and k, we reach a contradiction since such situations have been ruled out in Case 2. π' therefore lies on the lower envelope.

Consider the part $\gamma'_{\theta,c}$ of $\gamma_{\theta,c}$ that lies between (the last intersection of $\gamma_{\theta,c}$ with) π' and θ (as shown in Figure 4.2(b)). By construction, T_φ lies below Q_φ when we approach θ on $\gamma'_{\theta,c}$, but since π' lies on the lower envelope, T_φ lies above Q_φ when we approach π' on $\gamma'_{\theta,c}$. Thus T_φ has to intersect the relative interior of $\gamma'_{\theta,c}$, which implies the existence of a closed curve θ' defined by $T_\varphi \cap Q_\varphi$, distinct from θ, within level $\leq k$ (as illustrated in Figure 4.2(c)). As observed just before Lemma 4.1, $\gamma_{\theta,c}$ cannot intersect θ' at level 0. Thus the level of θ' is between 1 and k. Such a situation has however been ruled out in Case 2, leading to a contradiction.

We have shown that c can be charged by a connected component of an intersection of Q_φ with only one other surface. A symmetric statement holds for O_φ. $c \subseteq Q_\varphi \cap O_\varphi$ can thus be charged at most twice. □

We can now write the following relations for all $1 \leq j \leq q$. (Note that for $j = q$ the second term on the right side is not present.)

$$S^{(j)}(n) = O\left[k^3 S\left(\frac{n}{k}\right) + k^4 S^{(j+1)}\left(\frac{n}{k}\right) + k^6 C_{32}\left(\frac{n}{k}\right) + k^9 C_{32}\left(\frac{n}{k^2}\right) + k^4 n^{3+\varepsilon}\right]. \tag{4.8}$$

4.5. Putting it all together. We claim that the system of interdependent recurrences derived in this section, given in (4.4), (4.5), (4.7), (4.8), solves to

$$C_{32}(n) = O(n^{3+\varepsilon}), \qquad C_{22}(n) = O(n^{3+\varepsilon}), \qquad C_{222}(n) = O(n^{3+\varepsilon})$$

for any $\varepsilon > 0$. This is shown by induction, as in [20], choosing a different value of k for each recurrence. In more detail, we order the functions appearing in the recurrences as $(C_{22}, S^{(1)}, S^{(2)}, \ldots, S^{(q)}, C_{222}, C_{32})$ and denote this, for uniformity, as $(F_1, F_2, \ldots, F_{q+3})$. Each recurrence is roughly of the form

$$F_i(n) = O\left(k_i^{\beta_1} F_{j_1}\left(\frac{n}{k_i^{\alpha_1}}\right)\right) + O\left(k_i^{\beta_2} F_{j_2}\left(\frac{n}{k_i^{\alpha_2}}\right)\right) + \cdots + O\left(k_i^{\beta_r} F_{j_r}\left(\frac{n}{k_i^{\alpha_r}}\right)\right) + O\left(f_i(n)\right).$$

We represent this system by a directed graph G on the indices $\{1, 2, \ldots, q+3\}$, whose directed edges are $(i, j_1), \ldots, (i, j_r)$ for all i. We call an edge (i, j) a *forward* (resp., *backward*) edge if $j > i$ (resp., $j \leq i$). Let γ be the maximum of the ratios β_t/α_t taken over all corresponding edges (i, j_t) that are *backward* edges, and assume also that $f_i(n) = O(n^{\gamma+\varepsilon})$ for each i and for any $\varepsilon > 0$. Then one can show that the solution of this system is $F_i(n) = O(n^{\gamma+\varepsilon})$ for any $\varepsilon > 0$ and for all i. Informally, larger exponent ratios in terms that relate F_i to a function F_j with a *larger* index do not affect the overall bound because (almost all of) their effect can be suppressed by the choice of appropriate values for the k_i's, which decrease exponentially with i.

Since in our case, under the order given above, $\gamma = 3$, we obtain the bound asserted above.[2] This completes the proof of Theorem 1.1.

[2]Technically, γ is not quite 3 because of the factors $\beta_s(k)$ and k^ε that are also present in our recurrences. However, any $\gamma > 3$ can be used as a bound for the exponent of the solution, so $O(n^{3+\varepsilon})$ is a solution of the system.

5. Applications. Our bound on the complexity of overlays in $\mathbb{R}^4$ has many applications. We mention several of the more obvious ones. All the results listed below improve significantly upon the best previously known ones. Their proofs crucially rely on Theorem 1.1. Some of the more standard details in the proofs are omitted.

The region "sandwiched" between two envelopes. Let $\mathcal{F}$ and $\mathcal{G}$ be two families of n trivariate functions of constant description complexity, as above. Let $\Sigma_{\mathcal{F},\mathcal{G}}$ denote the *sandwich region* consisting of all points that lie above the upper envelope $E_{\mathcal{G}}$ of $\mathcal{G}$ and below the lower envelope $E_{\mathcal{F}}$ of $\mathcal{F}$. That is, $\Sigma_{\mathcal{F},\mathcal{G}}$ is the set of all quadruples (x_1, x_2, x_3, x_4), such that $g(x_1, x_2, x_3, x_4) \leq x_4 \leq f(x_1, x_2, x_3, x_4)$ for each $f \in \mathcal{F}$, $g \in \mathcal{G}$.

THEOREM 5.1. *The combinatorial complexity of the sandwich region $\Sigma_{\mathcal{F},\mathcal{G}}$ is $O(n^{3+\varepsilon})$ for any $\varepsilon > 0$.*

Proof. Consider, for example, the number of vertices of $\Sigma_{\mathcal{F},\mathcal{G}}$. Any such vertex is (i) a vertex of $E_{\mathcal{F}}$ or of $E_{\mathcal{G}}$ (there are $O(n^{3+\varepsilon})$ such vertices for any $\varepsilon > 0$), (ii) an intersection between an edge e of $E_{\mathcal{F}}$ and a facet φ of $E_{\mathcal{G}}$, (iii) an intersection between an edge e of $E_{\mathcal{G}}$ and a facet φ of $E_{\mathcal{F}}$, or (iv) an intersection between a 2-face f of $E_{\mathcal{F}}$ and a 2-face g of $E_{\mathcal{G}}$. Consider the overlay $Q(\mathcal{F}, \mathcal{G})$ of the minimization diagram $M_{\mathcal{F}}$ of $\mathcal{F}$ and the *maximization diagram* $M_{\mathcal{G}}$ of $\mathcal{G}$ (defined in complete analogy to the definition of minimization diagrams). In cases (ii) and (iii), the projections of e and of φ in $Q(\mathcal{F}, \mathcal{G})$ have a nonempty intersection. That is, there exists a connected portion of e that appears as a feature of $Q(\mathcal{F}, \mathcal{G})$, and the cells of $Q(\mathcal{F}, \mathcal{G})$ that it bounds are portions of the projection of φ. Similarly, in case (iv), the projections of f and of g intersect in a curve that is a feature (or a union of features) of $Q(\mathcal{F}, \mathcal{G})$. We can thus charge vertices of $\Sigma_{\mathcal{F},\mathcal{G}}$ to features of $Q(\mathcal{F}, \mathcal{G})$ in a unique manner, which clearly implies the claim. □

Note that the bound in Theorem 5.1 is nearly tight in the worst case. As a matter of fact, the proof of Theorem 5.1 implies the following stronger result; we refer the reader to [7, 17] for details concerning vertical decompositions in four dimensions.

COROLLARY 5.2. *The combinatorial complexity of the first stage of the vertical decomposition of the sandwich region $\Sigma_{\mathcal{F},\mathcal{G}}$ is $O(n^{3+\varepsilon})$ for any $\varepsilon > 0$.*

This corollary still leaves open the question of whether the complexity of the entire vertical decomposition of $\Sigma_{\mathcal{F},\mathcal{G}}$ is near-cubic or at least subquartic. This problem is still open even when one collection is empty, i.e., the problem concerning the vertical decomposition of the region below the lower envelope of a collection of trivariate functions.

The space of hyperplane transversals in 4-space. Let $\mathcal{C}$ be a collection of n convex sets in $\mathbb{R}^4$, each being semialgebraic of constant description complexity. Let $T_3(\mathcal{C})$ denote the space of all hyperplane transversals of $\mathcal{C}$, i.e., the set of all hyperplanes that intersect every member of $\mathcal{C}$. Using a standard duality transformation [10], we map hyperplanes to points and points to hyperplanes so that the incidence and the above/below relationships between points and hyperplanes are preserved. (This transformation excludes hyperplanes parallel to the x_4-axis, which can be handled separately in a much simpler manner.) Then each $c \in \mathcal{C}$ is mapped into two totally defined trivariate functions f_c^+, f_c^- such that a hyperplane $x_4 = h_1x_1+h_2x_2+h_3x_3+h_4$ intersects c if and only if $f_c^-(h_1, h_2, h_3) \leq h_4 \leq f_c^+(h_1, h_2, h_3)$. See [2] for more details. Hence $T_3(\mathcal{C})$ is the region sandwiched between the upper envelope of $\{f_c^- | c \in \mathcal{C}\}$ and the lower envelope of $\{f_c^+ | c \in \mathcal{C}\}$. Using Theorem 5.1, we thus obtain the following result.

THEOREM 5.3. *The combinatorial complexity of the space $T_3(\mathcal{C})$ of all hyperplane*

transversals of a set $\mathcal{C}$ of n convex sets of constant description complexity in $\mathbb{R}^4$ is $O(n^{3+\varepsilon})$ for any $\varepsilon > 0$.

Remark 3. Note that each vertex of $T_3(\mathcal{C})$ is dual to a hyperplane transversal that is tangent to four members of $\mathcal{C}$. Similar geometric interpretations hold for other features of $\partial T_3(\mathcal{C})$.

The space of line transversals in 3*-space.* Let $\mathcal{C}$ be a collection of n convex sets in $\mathbb{R}^3$, each being semialgebraic of constant description complexity. Let $T_1(\mathcal{C})$ denote the space of all line transversals of $\mathcal{C}$. We can map each line l in $\mathbb{R}^3$, given by the equations $y = a_1x + a_2$, $z = a_3x + a_4$, to the point $l^* = (a_1, a_2, a_3, a_4) \in \mathbb{R}^4$. (This excludes lines parallel to the yz-plane, which can be handled separately in a much simpler manner.) As above (see [2] for details), each $c \in \mathcal{C}$ is mapped to a pair of partially defined trivariate functions f_c^+, f_c^- such that f_c^+ and f_c^- have the same domain of definition, and a line l, with $l^* = (a_1, a_2, a_3, a_4)$, is a transversal of c if and only if the functions f_c^+, f_c^- are defined at (a_1, a_2, a_3) and $f_c^-(a_1, a_2, a_3) \le a_4 \le f_c^+(a_1, a_2, a_3)$. Hence this problem too reduces to a sandwich region in four dimensions, and Theorem 5.1 implies the following result.

THEOREM 5.4. *The combinatorial complexity of the space $T_1(\mathcal{C})$ of all line transversals of a set $\mathcal{C}$ of n convex sets of constant description complexity in $\mathbb{R}^3$ is $O(n^{3+\varepsilon})$ for any $\varepsilon > 0$.*

Remark 4. As above, vertices of $T_1(\mathcal{C})$ are dual to lines that are tangent to four members of $\mathcal{C}$.

This implies the following corollary concerning the number of *geometric permutations*. Such a permutation is the order in which a collection of disjoint convex bodies can be stabbed by a line transversal.

COROLLARY 5.5. *The number of geometric permutations in a collection $\mathcal{C}$ of n pairwise disjoint convex sets of constant description complexity in $\mathbb{R}^3$ is $O(n^{3+\varepsilon})$ for any $\varepsilon > 0$.*

This improves the general known upper bound of $O(n^4)$ [22] (for the special case of sets with constant description complexity) but is not known to be tight, since the only known lower bound is $\Omega(n^2)$ [16].

Efficient construction of transversal spaces. Theorems 5.3 and 5.4 do not address the problem of efficient construction of the respective spaces $T_3(\mathcal{C})$, $T_1(\mathcal{C})$. We next show that the boundaries of these spaces can be constructed efficiently in time $O(n^{3+\varepsilon})$ for any $\varepsilon > 0$. To avoid (the routine though somewhat technical) details involving the representation of the combinatorial structure of the boundary as a 3-dimensional cell complex, we restrict the algorithm to produce, for every pair $c_1, c_2 \in \mathcal{C}$, just the portion of $\partial T_3(\mathcal{C})$ (or $\partial T_1(\mathcal{C})$) that consists of all the points representing hyperplanes (or lines) that are tangent to c_1, c_2 and intersect all the other sets in $\mathcal{C}$. The representation of such a portion, which consists of some faces of a 2-dimensional arrangement (see below), is easy to define and compute. Our construction technique follows the approach used in [1, 5] and is easy to adapt to constructing a complete representation of $\partial T_3(\mathcal{C})$ (or $\partial T_1(\mathcal{C})$).

In more detail, let us consider the case of $T_3(\mathcal{C})$. Fix a pair of sets $c_1, c_2 \in \mathcal{C}$, and consider any pair of functions from $\{f_{c_1}^+, f_{c_1}^-\} \times \{f_{c_2}^+, f_{c_2}^-\}$, say, $f_{c_1}^+, f_{c_2}^+$. The intersection of their graphs is a 2-dimensional surface φ. For each $c \in \mathcal{C}^* \equiv \mathcal{C} \setminus \{c_1, c_2\}$, let

$$K_c = \{h \in \varphi \mid f_c^-(h_1, h_2, h_3) \le h_4 \le f_c^+(h_1, h_2, h_3)\}.$$

We need to construct $\bigcap_{c \in \mathcal{C}^*} K_c$. We do it using the randomized incremental technique of [1, 5]. That is, we insert the sets K_c for $c \in \mathcal{C}^*$ in some random order and update

the intersection after the insertion of each new set. We omit further details, which can be easily adapted from the algorithms just cited. The analysis given in [5], combined with Theorem 5.3, can easily be adjusted to the case at hand, implying that the overall expected running time of this algorithm, when applied to all pairs $c_1, c_2 \in \mathcal{C}$, is $O(n^{3+\varepsilon})$ for any $\varepsilon > 0$. Applying a fully analogous procedure for the case of line transversals in 3-space, we obtain the following theorem.

THEOREM 5.6. (a) *The boundary of the space of hyperplane transversals $T_3(\mathcal{C})$ in four dimensions, as defined above, can be computed in $O(n^{3+\varepsilon})$ randomized expected time for any $\varepsilon > 0$.*

(b) *The boundary of the space of line transversals $T_1(\mathcal{C})$ in three dimensions, as defined above, can be computed in $O(n^{3+\varepsilon})$ randomized expected time for any $\varepsilon > 0$.*

Union of objects in 4-space. Let $\mathcal{C}$ be a collection of n convex sets in $\mathbb{R}^4$, each being semialgebraic of constant description complexity, such that (i) the mean curvature [19] of any $c \in \mathcal{C}$ is at most some constant κ, and (ii) for any pair of sets $c_1, c_2 \in \mathcal{C}$, the ratio between their diameters is at most some constant α. (We refer to such sets as being of "nearly equal size.")

Let $\mathcal{U}$ denote the union of $\mathcal{C}$. The combinatorial complexity of $\mathcal{U}$ is the number of faces of all dimensions of the arrangement of the boundaries ∂c of the sets $c \in \mathcal{C}$, which appear on $\partial\mathcal{U}$.

THEOREM 5.7. *The combinatorial complexity of the union $\mathcal{U}$ of n convex sets of constant description complexity in $\mathbb{R}^4$ that satisfy properties* (i) *and* (ii) *is $O(n^{3+\varepsilon})$ for any $\varepsilon > 0$.*

Proof. We may assume that the diameter of any $c \in \mathcal{C}$ is between 1 and α. This, plus the bounded mean curvature assumption, implies the following two properties. Let G be an infinite axis-parallel grid in $\mathbb{R}^4$, where each cell of G is a hypercube of side length b for some sufficiently small constant $b < 1$. Then (a) each $c \in \mathcal{C}$ intersects only $O(1)$ cells of G; (b) let c be a set in $\mathcal{C}$ such that ∂c intersects a cell τ of G. Let $\Delta(c, \tau)$ denote the set of all directions d, such that $\partial c \cap \tau$ is monotone orthogonally to d. That is, $\partial c \cap \tau$ can be regarded as the graph of a (partially defined) function, where the dependent variable is in direction d. (Clearly, $\Delta(c, \tau)$ is centrally symmetric: $d \in \Delta(c, \tau) \Leftrightarrow -d \in \Delta(c, \tau)$.) Then the measure of $\Delta(c, \tau)$ is at least $7/8$ of the measure of the entire sphere of directions (provided b is sufficiently small).

This is easy to establish, and a similar analysis in three dimensions is provided in [4]. To verify property (b), note that the bounded mean curvature assumption implies that for any pair of points $x, y \in \partial c$ on a surface $c \in \mathcal{C}$

$$d_S(N_c(x), N_c(y)) \leq \kappa' \|x - y\|,$$

where κ' is a constant dependent on κ, $N_c(x)$ denotes the direction of the outward normal to ∂c at x, and d_S is the geodesic distance along the unit sphere of directions $\mathbb{S}^3$. Choose b sufficiently small so that $2\kappa' b < \delta$, where the value of δ will be determined shortly. Fix some $x_0 \in \partial c \cap \tau$. Let d be any direction forming an angle θ with $N_c(x_0)$. Suppose that there exists a line λ parallel to d that intersects $\partial c \cap \tau$ at two points u, v. Then $N_c(u) \cdot d$ and $N_c(v) \cdot d$ have different signs so that, say, $N_c(u) \cdot d < 0$. Assuming θ to be smaller than $\pi/2 - \delta$, it follows that $d_S(N_c(x_0), N_c(u)) \geq \pi/2 - \theta > \delta$. On the other hand, the bounded mean curvature assumption implies that $d_S(N_c(x_0), N_c(u)) \leq \kappa' \|u - x_0\| \leq 2\kappa' b \leq \delta$, which is a contradiction. This implies that the set of directions d that satisfy the property in (b) contains the two spherical caps centered at $\pm N_c(x_0)$ and having geodesic radius $\pi/2 - \delta$. Choosing δ (and b) sufficiently small, the validity of property (b) follows.

Property (b) implies that there exists a set Δ of $O(1)$ directions such that, for any quadruple $c_1, c_2, c_3, c_4 \in \mathcal{C}$ whose boundaries all cross a cell τ, we have $\Delta \cap \bigcap_{i=1}^{4} \Delta(c_i, \tau) \neq \emptyset$.

Now, fix a cell τ. If τ is fully contained in some set $c \in \mathcal{C}$, then we ignore τ; it contributes nothing to $\partial\mathcal{U}$. Otherwise, we consider the set $\mathcal{C}_\tau = \{c \in \mathcal{C} \mid \partial c \cap \tau \neq \emptyset\}$ and further partition it into the subsets $\mathcal{C}^+_{\tau,d}$, $\mathcal{C}^-_{\tau,d}$ for $d \in \Delta$, where $\mathcal{C}^+_{\tau,d}$ (resp., $\mathcal{C}^-_{\tau,d}$) consists of all sets $c \in \mathcal{C}_\tau$ for which $d \in \Delta(c, \tau)$ and such that, if we move slightly from any point on $\partial c \cap \tau$ in the direction $+d$ (resp., $-d$), we enter c. (A set c may belong to more than one of these subcollections.)

The preceding discussion implies the following property: (b') let $v \in \tau$ be a vertex of $\mathcal{U}$, incident to the boundaries of four sets $c_1, c_2, c_3, c_4 \in \mathcal{C}$. Then there exists a direction $d \in \Delta$ such that $c_1, c_2, c_3, c_4 \in \mathcal{C}^+_{\tau,d} \cup \mathcal{C}^-_{\tau,d}$.

Property (b') implies that the number of vertices of $\partial\mathcal{U} \cap \tau$ can be upper bounded by the sum, over $d \in \Delta$, of the number of vertices of the sandwich region between the upper envelope of the boundaries of the sets in $\mathcal{C}^+_{\tau,d}$ and the lower envelope of the boundaries of the sets in $\mathcal{C}^-_{\tau,d}$, where both boundaries are clipped to within τ. Using Theorem 5.1 plus the facts that $|\Delta| = O(1)$ and $\sum_\tau |\mathcal{C}_\tau| = O(n)$ (which follows from property (a)), the bound on the complexity of $\mathcal{U}$ follows. □

Arrangements with no vertices. The analysis in section 4.4, which uses a quite nonstandard counting scheme, is of interest on its own and can be adapted to other settings. In particular, it can be easily adjusted to show the following.

THEOREM 5.8. *The complexity of the lower envelope of an arrangement of n totally defined semialgebraic surfaces of constant description complexity in $\mathbb{R}^3$, that does not contain any vertices, is $O(n^{1+\varepsilon})$ for any $\varepsilon > 0$.*

Note that if the surfaces are not totally defined, the complexity of the lower envelope may still be quadratic. An easy lower bound construction is provided by a family of $n/2$ nearly x-parallel lines and another family of $n/2$ nearly y-parallel lines that together make up a grid-structure when viewed from below.

6. Conclusion. We have obtained several results concerning overlays of minimization diagrams using a novel approach. We feel that this approach might find applications for related problems like the analysis of vertical decompositions of arrangements [17]. Although the partition technique seems quite general, our initial steps in applying it in more general contexts have encountered some technical difficulties, which we hope to be able to resolve in the future. We also hope to be able to apply the technique to settle the conjecture concerning the complexity of overlays of minimization diagrams in all dimensions.

In general, it would be interesting to analyze the partition technique from a more "philosophical" point of view and to understand, in particular, the underlying reason for why it was so successful in the analysis of overlays, where the technique of counting schemes has provided only partial results but does not seem to be easily applicable to the related problem of the analysis of single cells, where a near-optimal solution using counting schemes exists [6]. The two techniques seem to be related, at least in the fact that the final recurrences that are derived by both techniques have very similar structures. This is apparent, for instance, in the case of overlays for bivariate functions, in which both techniques provide the same near-optimal solution. The difference, in this case at least, is that the way to obtain these recurrences via the partition technique is arguably much simpler.

Acknowledgments. Part of the work on this paper was carried out while the authors were visiting ETH Zürich, hosted by the European Graduate Program on Combinatorics, Geometry and Computation. The authors are grateful to the program, and, in particular, to Emo Welzl, for the hospitality. Emo Welzl also contributed valuable ideas through helpful discussions.

REFERENCES

[1] P. K. Agarwal, B. Aronov, and M. Sharir, *Computing envelopes in four dimensions with applications*, SIAM J. Comput., 26 (1997), pp. 1714–1732.

[2] P. K. Agarwal, O. Schwarzkopf, and M. Sharir, *The overlay of lower envelopes and its applications*, Discrete Comput. Geom., 15 (1996), pp. 1–13.

[3] P. K. Agarwal and M. Sharir, *Arrangements and their applications*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., North–Holland, Amsterdam, 2000, pp. 49–119.

[4] P. K. Agarwal and M. Sharir, *Pipes, cigars, and kreplach: The union of Minkowski sums in three dimensions*, Discrete Comput. Geom., 24 (2000), pp. 645–685.

[5] B. Aronov, M. Sharir, and B. Tagansky, *The union of convex polyhedra in three dimensions*, SIAM J. Comput., 26 (1997), pp. 1670–1688.

[6] S. Basu, *On the combinatorial and topological complexity of a single cell*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 606–616.

[7] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir, *A singly-exponential stratification scheme for real semi-algebraic varieties and its applications*, Theoret. Comput. Sci., 84 (1991), pp. 77–105.

[8] K. L. Clarkson and P. W. Shor, *Applications of random sampling in computational geometry* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[9] M. de Berg and O. Schwarzkopf, *Cuttings and applications*, Internat. J. Comput. Geom. Appl., 5 (1995), pp. 343–355.

[10] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, EATCS Monographs on Theoretical Computer Science 10, Springer-Verlag, Berlin, 1987.

[11] H. Edelsbrunner, L. J. Guibas, and M. Sharir, *The upper envelope of piecewise linear functions: Algorithms and applications*, Discrete Comput. Geom., 4 (1989), pp. 311–336.

[12] D. Halperin and M. Sharir, *New bounds for lower envelopes in three dimensions, with applications to visibility in terrains*, Discrete Comput. Geom., 12 (1994), pp. 313–326.

[13] D. Halperin and M. Sharir, *Almost tight upper bounds for the single cell and zone problems in three dimensions*, Discrete Comput. Geom., 14 (1995), pp. 385–410.

[14] S. Har-Peled, *The Complexity of Many Cells in the Overlay of Many Arrangements*, M.Sc. dissertation, School of Computer Science, Tel Aviv University, Tel Aviv, Israel, 1995.

[15] S. Har-Peled, *Multicolor combination lemma*, Comput. Geom., 12 (1999), pp. 155–176.

[16] M. Katchalski, T. Lewis, and A. Liu, *The different ways of stabbing disjoint convex sets*, Discrete Comput. Geom., 7 (1992), pp. 197–206.

[17] V. Koltun, *Almost tight upper bounds for vertical decompositions in four dimensions*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 56–65.

[18] V. Koltun and M. Sharir, *On the overlay of envelopes in four dimensions*, in Proceedings of the 13th ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2002, pp. 810–819.

[19] A. Pressley, *Elementary Differential Geometry*, Springer-Verlag, London, 2001.

[20] M. Sharir, *Almost tight upper bounds for lower envelopes in higher dimensions*, Discrete Comput. Geom., 12 (1994), pp. 327–345.

[21] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, Cambridge, UK, 1995.

[22] R. Wenger, *Upper bounds on geometric permutations for convex sets*, Discrete Comput. Geom., 5 (1990), pp. 27–33.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 864–879

COMPUTING PHYLOGENETIC ROOTS WITH BOUNDED DEGREES AND ERRORS*

ZHI-ZHONG CHEN†, TAO JIANG‡, AND GUOHUI LIN§

Abstract. Given a set of species and their similarity data, an important problem in evolutionary biology is how to reconstruct a phylogeny (also called evolutionary tree) so that species are close in the phylogeny if and only if they have high similarity. Assume that the similarity data are represented as a graph $G = (V, E)$, where each vertex represents a species and two vertices are adjacent if they represent species of high similarity. The phylogeny reconstruction problem can then be abstracted as the problem of finding a (phylogenetic) tree T from the given graph G such that (1) T has no degree-2 internal nodes, (2) the external nodes (i.e., leaves) of T are exactly the elements of V, and (3) $(u, v) \in E$ if and only if $d_T(u, v) \leq k$ for some fixed threshold k, where $d_T(u, v)$ denotes the distance between u and v in tree T. This is called the *phylogenetic kth root problem* (PRk), and such a tree T, if it exists, is called a *phylogenetic kth root* of graph G. The computational complexity of PRk is open, except for $k \leq 4$. In this paper, we investigate PRk under a natural restriction that the maximum degree of the phylogenetic root is bounded from above by a constant. Our main contribution is a linear-time algorithm that determines if G has such a phylogenetic kth root, and if so, demonstrates one. On the other hand, because in practice the collected similarity data are usually not perfect and may contain errors, we propose to study a generalized version of PRk where the output phylogeny is required only to be an *approximate* root of the input graph. We show that this and other related problems are computationally intractable.

Key words. phylogeny, phylogenetic root, computational biology, efficient algorithm, NP-hard

AMS subject classifications. 05C05, 05C85, 05C90, 68R10, 90C27, 94C30

DOI. 10.1137/S0097539701389154

1. Introduction.

The reconstruction of evolutionary history for a set of species from quantitative biological data has long been a popular problem in computational biology. This evolutionary history is typically modeled by an evolutionary tree, or *phylogeny*. A phylogeny is a tree where the leaves are labeled by species and each internal node represents a speciation event whereby an ancestral species gives rise to two or more child species. Both rooted and unrooted trees have been used to describe phylogenies in the literature, although they are practically equivalent. In this paper, we will consider only unrooted phylogenies for the convenience of presentation.[1] The internal nodes of a phylogeny have degrees (in the sense of unrooted trees, i.e.,

*Received by the editors May 8, 2001; accepted for publication (in revised form) March 11, 2003; published electronically June 10, 2003. An extended abstract of this paper appeared in *Proceedings of the Seventh International Workshop on Algorithms and Data Structure* (WADS 2001), Lecture Notes in Comput. Sci. 2125, Springer-Verlag, Berlin, 2001, pp. 377–388.

http://www.siam.org/journals/sicomp/32-4/38915.html

†Department of Mathematical Sciences, Tokyo Denki University, Hatoyama, Saitama 350-0394, Japan (chen@r.dendai.ac.jp). This work was done while this author was visiting the Department of Computer Science, University of California, Riverside, and was supported in part by the Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan under grant 12780241.

‡Department of Computer Science, University of California, Riverside, CA 92521 (jiang@cs.ucr.edu). This author was supported in part by a UCR startup grant and NSF grants CCR-9988353 and ITR-0085910.

§Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada (ghlin@cs.ualberta.ca). This author's work was performed mostly at McMaster University and the University of Waterloo and was supported in part by NSERC and a Startup Grant from the University of Alberta.

[1]But some of our hardness proofs will also use rooted trees as intermediate data structures in the construction.

the number of incident edges) at least 3. Proximity within a phylogeny in general corresponds to similarity in evolutionary characteristics.

Many phylogenetic reconstruction algorithms have been proposed and studied in the literature [12]. In this paper we investigate the computational feasibility of a graph-theoretic approach for reconstructing phylogenies from similarity data. Specifically, interspecies similarity is represented by a graph where the vertices are the species and the adjacency relation represents evidence of evolutionary similarity. A phylogeny is then reconstructed from the graph such that the leaves of the phylogeny are labeled by vertices of the graph (i.e., species) and for any two vertices of the graph, they are adjacent in the graph if and only if their corresponding leaves in the phylogeny are connected by a path of length at most k, where k is a predetermined proximity threshold. To be clear, vertices in the graph are called *vertices* while those in the phylogeny are called *nodes*. Recall that the length of the (unique) path connecting two nodes u and v in phylogeny T is the number of edges on the path, which is denoted by $d_T(u, v)$. This approach gives rise to the following algorithmic problem [8].

PHYLOGENETIC kTH ROOT PROBLEM (PRk). *Given a graph $G = (V, E)$, find a phylogeny T with leaves labeled by the elements of V such that for each pair of vertices $u, v \in V$, $(u, v) \in E$ if and only if $d_T(u, v) \leq k$.*

Such a phylogeny T (if it exists) is called a *phylogenetic kth root*, or a *kth root phylogeny*, of graph G. Graph G is called the *kth phylogenetic power* of T. For convenience, we denote the kth phylogenetic power of any phylogeny T as T^k. That is, $T^k = \{(u, v) \mid u \text{ and } v \text{ are leaves of } T \text{ and } d_T(u, v) \leq k\}$. Thus, PR$k$ asks for a phylogeny T such that $G = T^k$.

1.1. Connection to graph and tree roots, and previous results. Phylogenetic power might be thought of as a *Steiner* extension of the standard notion of *graph power*. A graph G is the *kth power* of a graph H (or equivalently, H is a *kth root* of G) if vertices u and v are adjacent in G if and only if the length of the shortest path from u to v in H is at most k. An important special case of graph power/root problems is the following.

TREE kTH ROOT PROBLEM (TRk). *Given a graph $G = (V, E)$, find a tree $T = (V, E_T)$ such that $(u, v) \in E$ if and only if $d_T(u, v) \leq k$.*

If T exists, then it is called a *tree kth root*, or a *kth root tree*, of graph G. Note that in the phylogenetic root problem the leaves of T correspond to the vertices of G, while in the tree root problem the nodes of T correspond to the vertices of G.

The special case TR2 is also known as the *tree square root problem* [9]. Correspondingly, we call PR2 the *phylogenetic square root problem*. There is abundant literature on graph root and power (see [2, section 10.6] for an overview) but few results on phylogenetic/tree roots/powers. It is NP-complete to recognize a graph power [10]; nonetheless, it is possible to determine if a graph has a kth root tree, for any fixed k, in $O(n^3)$ time, where n is the number of vertices in the input graph [5]. In particular, determining if a graph has a tree square root can be done in $O(n + e)$ time [9], where e is the number of edges in the input graph. Recently, Nishimura, Ragde, and Thilikos [11] presented an $O(n^3)$-time algorithm for a variant of PRk for $k \leq 4$, where internal nodes of the output phylogeny are allowed to have degree 2. More recently, Lin, Kearney, and Jiang [8] introduced a novel notion of *critical clique* and obtained an $O(n + e)$-time algorithm for PRk for $k \leq 4$. Unfortunately, both algorithms cannot be generalized to $k \geq 5$.

1.2. Our contribution. In the practice of phylogeny reconstruction, most phylogenies considered are trees of degree 3 [12] because speciation events are usually

bifurcating events in the evolutionary process. In such *fully resolved* phylogenetic trees, each internal node has three neighbors and represents a speciation event in which some ancestral species splits into two child species. Nodes of degrees higher than 3 are introduced only when the input biological (similarity) data is not sufficient to separate individual speciation events, and hence several such events may be collapsed into a nonbifurcating (super) speciation event in the reconstructed phylogeny. Hence in this paper, we consider a restricted version of PRk where the output phylogeny is assumed to have degree at most Δ for some fixed constant $\Delta \geq 3$. For simplicity, we call it the *degree-*Δ PRk and denote it in short as ΔPRk. Since in the practice of computational biology the species under consideration are more or less related, we are mostly interested in connected graphs. The main contribution of this paper is a linear-time algorithm that determines, for any input connected graph G and constant $\Delta \geq 3$, if G has a kth root phylogeny with degree at most Δ, and if so, demonstrates one such phylogeny. The basic construction in our algorithm is a nontrivial application of bounded-width tree-decomposition of certain chordal graphs [2].

Notice that the input graph in PRk is derived from some similarity data, which is usually inexact in practice and may have erroneous (spurious or missing) edges. Such errors may result in graphs that have no phylogenetic roots. Hence, it is natural to consider a more relaxed problem where we look for phylogenetic trees whose powers are *close* to the input graphs. The precise formulation is as follows.

CLOSEST PHYLOGENETIC kTH ROOT PROBLEM (CPRk). *Given a graph* $G = (V, E)$ *and a nonnegative integer* ℓ*, find a phylogeny* T *with leaves labeled by* V *such that* G *and* T^k *differ by at most* ℓ *edges. That is,*

$$\left|E(G) \oplus E(T^k)\right| = \left|\left(E(G) - E(T^k)\right) \cup \left(E(T^k) - E(G)\right)\right| \leq \ell.$$

A phylogeny T which minimizes the above edge discrepancy is called a *closest* k*th root phylogeny* of graph G.

The *closest tree* k*th root problem* (CTRk) is defined analogously. Notice that CTR1 is trivially solved by finding a spanning tree of the input graph. Kearney and Corneil [5] proved that CTRk is NP-complete when $k \geq 3$. The computational complexity for CTR2 had been open for a while and was recently shown to be intractable by Jiang, Lin, and Xu [4]. In this paper, we will show that CPRk is NP-complete for any $k \geq 2$. Another closely related problem, the *Steiner* k*th root problem* (where $k \geq 1$), is also studied.

We introduce some notation and definitions, as well as some existing related results, in the next section. Our main result on bounded-degree PRk is presented in section 3. The hardness of closest phylogenetic root and Steiner root problems is discussed in section 4. We close the paper with some open problems in section 5.

2. Preliminaries. We employ standard terminologies in graph theory. In particular, the subgraph of a graph G induced by a vertex set U of G is denoted by $G[U]$, the degree of a vertex v in G is denoted by $deg_G(v)$, and the maximum size of a clique in G is denoted by $\omega(G)$. First, it is obvious that if a graph has a kth root phylogeny, then it must be *chordal*; that is, it contains no induced subgraph which is a cycle of size greater than 3 [2].

DEFINITION 2.1. *A* tree-decomposition *of a graph* $G = (V, E)$ *is a pair* $\mathcal{D} = (\mathcal{T}, \mathcal{B})$ *consisting of a tree* $\mathcal{T} = (U, F)$ *and a collection* $\mathcal{B} = \{B_\alpha \mid B_\alpha \subseteq V, \alpha \in U\}$ *of* sets *(called* bags*) for which*

- $\bigcup_{\alpha \in U} B_\alpha = V$;

- *for each edge* $(v_1, v_2) \in E$*, there is a node* $\alpha \in U$ *such that* $\{v_1, v_2\} \subseteq B_\alpha$*; and*
- *if* $\alpha_2 \in U$ *is on the path connecting* α_1 *and* α_3 *in* $\mathcal{T}$*, then* $B_{\alpha_1} \cap B_{\alpha_3} \subseteq B_{\alpha_2}$.

The treewidth *associated with this tree-decomposition* $\mathcal{D} = (\mathcal{T}, \mathcal{B})$ *is* $tw(G, \mathcal{D}) = \max_{\alpha \in U} |B_\alpha| - 1$.

The treewidth *of graph* G*, denoted by* $tw(G)$*, is the minimum* $tw(G, \mathcal{D})$ *taken over all tree-decompositions* $\mathcal{D}$ *of* G.

A clique-tree-decomposition *of* G *is a tree-decomposition* $(\mathcal{T}, \mathcal{B})$ *of* G *such that each bag in* $\mathcal{B}$ *is a maximal clique of* G.

LEMMA 2.2 (see [6]). *Every chordal graph has a clique-tree-decomposition, and it can be computed in linear time.*

From the proof of Lemma 2.2 given in [6], it is not difficult to see that a clique-tree-decomposition $\mathcal{D} = (\mathcal{T}, \mathcal{B})$ of a given chordal graph G can be computed in linear time if $\omega(G) = O(1)$. We can further modify $\mathcal{D}$ so that $deg_{\mathcal{T}}(\alpha) \leq 3$ for each node α of $\mathcal{T}$ [1]. This modification takes linear time too if $\omega(G) = O(1)$.

Hereafter, a tree-decomposition of a chordal graph G always means a clique-tree-decomposition $\mathcal{D} = (\mathcal{T}, \mathcal{B})$ of G such that $deg_{\mathcal{T}}(\alpha) \leq 3$ for all nodes α of $\mathcal{T}$. Furthermore, in what follows, we abuse the notation to use $\mathcal{D}$ to denote the tree $\mathcal{T}$ in it (since we will use T to denote the kth root phylogeny of graph G) and denote the bag associated with a node α of $\mathcal{D}$ by B_α.

3. Algorithm for bounded-degree PR*k*. This section presents a linear-time algorithm for solving 3PRk. The adaptation to ΔPRk where $\Delta \geq 4$ is straightforward and hence omitted here.

We assume that the input graph $G = (V, E)$ is connected. We further assume that G is not complete but is chordal; otherwise the problem is trivially solved in linear time. Since every vertex $v \in V$ appears as an external node (i.e., leaf) in the kth root phylogeny, the maximum size $\omega(G)$ of a clique in G can be bounded from above by a constant $f(k)$, where

$$f(k) = \begin{cases} 3 \cdot 2^{\frac{k}{2}-1} & \text{if } k \text{ is even,} \\ 2^{\frac{k+1}{2}} - 1 & \text{if } k \text{ is odd.} \end{cases}$$

The first step of the algorithm consists of constructing a clique-tree-decomposition $\mathcal{D}$ of G in linear time, as given in Lemma 2.2. Then the kth root phylogeny is computed by a dynamic programming algorithm applied on a rooted version of the decomposition $\mathcal{D}$. The dynamic programming starts at the leaves of $\mathcal{D}$ and proceeds upwards. After processing the root, the algorithm will construct a kth root phylogeny of G if there is any. The processing of a node α of $\mathcal{D}$ can be sketched as follows. Let U_α be the union of the bags associated with α and its descendants in $\mathcal{D}$. While processing α, the algorithm computes a set of trees T such that (1) T may possibly be a subtree of a kth root phylogeny of G, (2) all vertices of U_α are leaves of T, and (3) each leaf of T not contained in U_α is not labeled. The unlabeled leaves of T serve as ports from which we can expand T so that it may eventually become a kth root phylogeny of G. The crucial point we will observe is that we need only those ports that are at distance at most k apart from vertices of B_α in T. This point implies that the number of necessary ports depends only on k and hence is a constant. The details of the dynamic programming algorithm are developed in the next two subsections.

One more notation is in order. For two adjacent nodes α and β of $\mathcal{D}$, let $U(\alpha, \beta) = \bigcup_\gamma B_\gamma$, where γ ranges over all nodes of $\mathcal{D}$ with $d_{\mathcal{D}}(\gamma, \alpha) < d_{\mathcal{D}}(\gamma, \beta)$. In other words,

if we root $\mathcal{D}$ at node β, then $U(\alpha, \beta)$ is the union of the bags associated with α and its descendants in $\mathcal{D}$. A useful property of $\mathcal{D}$ is that for every internal node β and every two neighbors α_1 and α_2 of β in $\mathcal{D}$, G has no edge between any vertex of $U(\alpha_1, \beta) - B_\beta$ and any vertex of $U(\alpha_2, \beta) - B_\beta$.

3.1. Ideas behind the dynamic programming algorithm. Note that since $\Delta = 3$, every internal node in a kth root phylogeny T of G has degree exactly 3.

DEFINITION 3.1. *Let U be a set of vertices of G. A* relaxed phylogeny *for U is a tree R satisfying the following conditions:*

- *The degree of each internal node in R is* 3.
- *Each vertex of U is a leaf in R and appears in R only once. For convenience, we call the leaves of R that are also vertices of U* final leaves *of R, and we call the rest of the leaves of R* temporary leaves *of R.*
- *For every two vertices u and v of U, u and v are adjacent in G if and only if $d_R(u, v) \leq k$.*
- *Each temporary leaf v of R is assigned a pair (γ, t), where γ is a node of $\mathcal{D}$ and $0 \leq t \leq k$. We call γ the* color *of v and call t the* threshold *of v. For convenience, we denote the color of a temporary leaf v of R by $c_R(v)$, and denote the threshold of v by $t_R(v)$.*

Intuitively speaking, the temporary leaves of R serve as ports from which we can expand R so that it may eventually become a kth root phylogeny of G.

Recall that our algorithm processes the nodes of $\mathcal{D}$ one by one. While processing a node α of $\mathcal{D}$, the algorithm finds out all relaxed phylogenies for B_α that are subtrees of kth root phylogenies of graph G. The following lemma shows that such relaxed phylogenies for B_α have certain useful properties.

LEMMA 3.2. *Let T be a kth root phylogeny of G. Let α be a node of $\mathcal{D}$. Root T at an arbitrary leaf that is in B_α. Define a* pure *node to be a node w of T such that α has a neighbor γ in $\mathcal{D}$ such that all leaf descendants of w in T are in $U(\gamma, \alpha) - B_\alpha$. Define a* critical *node to be a pure node of T whose parent is not pure. Let R be the relaxed phylogeny for B_α obtained from T by performing the following steps of operations:*

1. *For every critical node w of T, perform the following:*
   (a) *Compute the minimum distance from w to a leaf descendant of w in T; let i_w denote this distance. (*Comment: *$i_w \leq k$ or else the leaf descendants of w in tree T would be unreachable from the outside in graph G.)*
   (b) *Find the neighbor γ of α such that all leaf descendants of w in T are contained in $U(\gamma, \alpha)$.*
   (c) *Delete all descendants (excluding w, of course) of w, and assign the pair (γ, i_w) to w.*
2. *Unroot T.*

Then the resultant R has the following properties:

- *For every temporary leaf v of R, $c_R(v)$ is a neighbor of α in $\mathcal{D}$.*
- *For every two temporary leaves u and v of R with different colors, it holds that $t_R(u) + t_R(v) + d_R(u, v) > k$.*
- *For every neighbor γ of α in $\mathcal{D}$, every temporary leaf v of R with $c_R(v) = \gamma$, and every final leaf w of R with $w \notin B_\gamma$, it holds that $d_R(v, w) + t_R(v) > k$.*
- *For every internal node v of R, either there is a final leaf u of R with $d_R(u, v) \leq k - 1$ or at least one descendant of v in R' is a final leaf of R, where R' is obtained from R by rooting it at an arbitrary final leaf.*

Proof. Obviously, R is a relaxed phylogeny for B_α. Since T is a phylogeny of G, it follows immediately that R has the first three properties in the lemma. To prove the fourth property, first note that the construction of R from T is independent of the choice of the root of T as long as it is a final leaf. So, it suffices to prove that for every critical node w of T whose parent p in T has no leaf descendant contained in B_α, there is a vertex u in B_α such that $d_T(u,p) \leq k-1$. To this end, let v be a leaf descendant of p that is closest to p among all leaf descendants of p in T. Let $u_1, u_2, \ldots, u_q$ be all leaves in T that are at distance at most k apart from v in T but are not descendants of p in T. Since G is connected, $q \geq 1$. We claim that $\{u_1, u_2, \ldots, u_q\} \cap B_\alpha \neq \emptyset$. For the sake of a contradiction, assume that $\{u_1, u_2, \ldots, u_q\} \cap B_\alpha = \emptyset$. Then some connected component G_1 of $G[V - B_\alpha]$ contains all of $v, u_1, u_2, \ldots, u_q$. Let Q be the set of all leaf descendants of p in T that are not contained in G_1. Since p is not pure and no leaf descendant of p in T is in B_α, we have $|Q| \geq 1$. Also, no vertex of G_1 can be adjacent to any vertex of Q in G. Now, by the choices of u_1 through u_q and the assumption that G_1 contains all of $u_1, u_2, \ldots, u_q$, we conclude that G has no edge between any vertex of Q and any vertex of $V - Q$. This contradicts the connectivity of G. So, the claim holds. By this claim, there is a $u_i \in \{u_1, u_2, \ldots, u_q\} \cap B_\alpha$ such that $d_T(u_i, v) \leq k$. Thus, $d_T(u_i, p) \leq k-1$, establishing the fourth property. ∎

Each relaxed phylogeny R for B_α having the four properties in Lemma 3.2 is called a *skeleton* of α. The following lemma shows that there can be only a constant number of skeletons of α.

LEMMA 3.3. *For each node α of $\mathcal{D}$, the number of skeletons of α is bounded from above by a constant depending only on k and $|B_\alpha|$.*

Proof. First note that the color and the threshold of each temporary leaf can be chosen from a constant range. Further note that each internal node v in a skeleton S of α satisfies $deg_S(v) = 3$. So, to prove the lemma, it suffices to prove that the number of temporary leaves in a skeleton S of α is bounded from above by a constant.

Consider a skeleton S of α and root S at an arbitrary final leaf r. We claim that for every temporary leaf u of S, there is a final leaf w with $d_S(u,w) \leq k$. To see this, let u be a temporary leaf and v be the parent of u in S. Since the root of S is a final leaf, v must be an internal node of S. If there is a final leaf w with $d_S(v,w) \leq k-1$, then $d_S(u,w) \leq k$ and we are done. Otherwise, by the definition of a skeleton, at least one descendant x of v is a final leaf of S. Since $d_S(x,r) \leq k$, we have $\min\{d_S(x,v), d_S(r,v)\} \leq k-1$ and hence $\min\{d_S(x,u), d_S(r,u)\} \leq k$. This establishes the claim.

Since each pair of final leaves is at distance at most k apart in S and the maximum degree of a node in S is 3, there are only a constant number of temporary leaves by the claim. ∎

By Lemma 3.3, while processing a node α of $\mathcal{D}$, our algorithm can find out all skeletons of α in constant time. For each skeleton S of α, if possible, the algorithm then extends S to a relaxed phylogeny for $U(\alpha, \beta)$, where β is the parent of α in rooted $\mathcal{D}$. The algorithm records these relaxed phylogenies of α in the dynamic programming table for later use when processing the parent β. The following definition aims at removing unnecessary relaxed phylogenies of α from the dynamic programming table.

DEFINITION 3.4. *Let α and β be two adjacent nodes of $\mathcal{D}$. Let S be a skeleton of α. The* projection *of S to β is a relaxed phylogeny for $B_\alpha \cap B_\beta$ obtained from S by performing the following steps of operations:*

1. *Change each final leaf $v \notin B_\beta$ to a temporary leaf; set the threshold of v to be 0 and set the color of v to be α.*

2. *Root S at an arbitrary vertex of $B_\alpha \cap B_\beta$.*
3. *Find those nodes v in S such that* (i) *every leaf descendant of v in S is a temporary leaf whose color is not β, but* (ii) *the parent of v in S does not have property* (i).
4. *For each node v found in the last step, if v is a leaf in S, then set the color of v to be α; otherwise, perform the following steps of operations:*
   (a) *Set $m_v = \min_u\{t_S(u) + d_S(u, v)\}$, where u ranges over all leaf descendants of v in S.*
   (b) *Delete all descendants of v in S.*
   (c) *Set v to be a temporary leaf, α to be the color of v, and m_v to be the threshold of v.*
5. *Unroot S.*

Obviously, two different skeletons of α may have the same projection to β. For convenience, we say that these skeletons are *equivalent*. Among equivalent skeletons of α, our algorithm will extend only a hopeful one of them to a relaxed phylogeny for $U(\alpha, \beta)$ and record it in the dynamic programming table. This motivates the following definition.

DEFINITION 3.5. *Let α and β be two adjacent nodes of $\mathcal{D}$. A* projection *of α to β is the projection of a skeleton of α to β. Let P be a projection of α to β. An* expansion *of P to $U(\alpha, \beta)$ is a relaxed phylogeny X for $U(\alpha, \beta)$ such that some subtree Y of X is isomorphic to P, and the bijection f from the node set of P to the node set of Y witnessing this isomorphism satisfies the following conditions:*

- *For every final leaf v of P, $f(v) = v$.*
- *For every temporary leaf v of P with $c_P(v) = \beta$, $f(v)$ is a temporary leaf of X with $c_X(f(v)) = c_P(v)$ and $t_X(f(v)) = t_P(v)$.*
- *Suppose that we root X at a vertex in $B_\alpha \cap B_\beta$. Then, for every temporary leaf v of P with $c_P(v) \neq \beta$ (hence $c_P(v) = \alpha$), all leaf descendants of $f(v)$ in X are final leaves and are contained in $U(\alpha, \beta) - B_\beta$, and the minimum distance between $f(v)$ and a leaf descendant of $f(v)$ in X equals $t_P(v)$.*

Note that a projection of α to β may have no expansion to $U(\alpha, \beta)$. The following lemma shows that if G has a kth root phylogeny T, then some subtree of T is a projection of α to β and another subtree of T is its expansion to $U(\alpha, \beta)$.

LEMMA 3.6. *Let α and β be two adjacent nodes in $\mathcal{D}$. Let T be a kth root phylogeny of G. Root T at an arbitrary leaf that is in B_α. Let R be the skeleton of α obtained from T as in Lemma* 3.2. *Let P be the projection of R to β. Define a* β-pure *node to be a node w of T such that all leaf descendants of w in T are contained in $U(\beta, \alpha) - B_\alpha$. Further, define a* β-critical *node to be a β-pure node of T whose parent is not β-pure. Let X be the relaxed phylogeny for $U(\alpha, \beta)$ obtained from T by performing the following steps of operations:*

1. *For every β-critical node w of T, perform the following:*
   (a) *Compute the minimum distance from w to a leaf descendant of w in T; let i_w denote this distance. (*Comment: *$i_w \leq k$ or else the leaf descendants of w in tree T would be unreachable from the outside in graph G.)*
   (b) *Delete all descendants (excluding w, of course) of w, and assign the pair (β, i_w) to w.*
2. *Unroot T.*

Then X is an expansion of P to $U(\alpha, \beta)$.

Proof. The proof is straightforward. ∎

By Lemma 3.6, whenever G has a kth root phylogeny, there is always a projection of α to β that has an expansion to $U(\alpha, \beta)$. While processing α, our algorithm will find out those projections that have expansions to $U(\alpha, \beta)$ and record the expansions in the dynamic programming table.

3.2. Details of dynamic programming for 3PR*k*. To solve the 3PRk problem for G, we perform a dynamic programming on the tree-decomposition $\mathcal{D}$ as follows. To simplify the description of the algorithm, we add a new node r to $\mathcal{D}$, connect r to an arbitrary leaf α of $\mathcal{D}$, and copy the bag at α to r (that is, $B_r = B_\alpha$). Clearly, the resultant $\mathcal{D}$ is still a required tree-decomposition of G. Root $\mathcal{D}$ at r.

The dynamic programming starts at the leaves of $\mathcal{D}$ and proceeds upwards; after the unique child of the root r of $\mathcal{D}$ is processed, we will know whether G has a kth root phylogeny or not. The invariant maintained during the dynamic programming is that after each nonroot node α has been processed, for each projection P of α to its parent β, we will have found out whether P has an expansion X to $U(\alpha, \beta)$, and we will have found and recorded such an X if any exists.

Now consider how a nonroot node α of $\mathcal{D}$ is processed. Let β be the parent of α in $\mathcal{D}$. First suppose that α is a leaf of $\mathcal{D}$. When processing α, we find and record all possible projections of α to β; moreover, for each projection P found, we also record a skeleton S of α such that P is the projection of S to β.

Next suppose that α is neither a leaf nor the root node of $\mathcal{D}$, and suppose that all descendants of α in $\mathcal{D}$ have been processed. To process α, we try all possible skeletons S of α. When trying S, for each child γ of α in $\mathcal{D}$, we first compute the projection P_γ of S to γ and then check whether P_γ is also a projection of γ to α and additionally has an expansion to $U(\gamma, \alpha)$. If the checking fails for at least one child of α, we proceed to try the next possible skeleton of α. Otherwise, we can conclude that the projection P_β of S to β has an expansion to $U(\alpha, \beta)$ because such an expansion can be constructed as follows:

1. For each child γ of α in $\mathcal{D}$, search the dynamic programming table to find the expansion X_γ of P_γ to $U(\gamma, \alpha)$, and find the bijection f_γ (from the node set of P_γ to the node set of some subtree of X_γ) witnessing that X_γ is an expansion of P_γ to $U(\gamma, \alpha)$. (*Comment:* To speed up the algorithm, we may have recorded this bijection in the dynamic programming table when processing γ.)
2. For each child γ of α in $\mathcal{D}$, root X_γ at an arbitrary vertex of $B_\gamma \cap B_\alpha$.
3. Modify S as follows: For each temporary leaf v of S with $c_S(v) \neq \beta$, replace v by the subtree rooted at $f_\gamma(v)$ of X_γ, where $\gamma = c_S(v)$. (*Comment:* Recall that by Definition 3.4 each temporary leaf v of S with $c_S(v) = \gamma$ is also a temporary leaf of P_γ.)

One can verify that the above construction indeed gives us an expansion of P_β. Since P_β is a possible projection of α to β, we record this expansion for P_β in the dynamic programming table. After trying all possible skeletons of α, if we find no projection of α to β that has an expansion to $U(\alpha, \beta)$, then we can conclude that G has no kth root phylogeny; otherwise, we proceed to the processing of the next node of $\mathcal{D}$.

Finally, suppose that α is the unique child of the root r of $\mathcal{D}$. Further suppose that α has been successfully processed (for otherwise we already knew that G has no kth root phylogeny). Then, by searching the dynamic programming table, we try to find a projection P of α to r such that (i) P has no temporary leaf whose color is r, and (ii) an expansion X of P to $U(\alpha, r)$ has been recorded in the dynamic programming table. If P is found, we can conclude that X is a kth root phylogeny for G; otherwise,

we can conclude that G has no kth root phylogeny.

The above discussion justifies the following theorem.

THEOREM 3.7. *Let k be a constant integer larger than or equal to 2. There is a linear-time algorithm determining if a given connected graph has a kth root phylogeny in which every internal node has degree 3, and if so, demonstrating one such phylogeny.*

We can easily generalize the above discussion to prove the following.

COROLLARY 3.8. *Let Δ and k be constant integers such that $\Delta \geq 3$ and $k \geq 2$. There is a linear-time algorithm determining if a given connected graph has a kth root phylogeny in which every internal node has degree in the range $[3, \Delta]$, and if so, demonstrating one such phylogeny.*

4. The hardness of closest phylogenetic root problems. We introduce some basic concepts (some of them from [5]) that will be used in the hardness proofs. Consider a set $S = \{s_1, s_2, \ldots, s_n\}$. Let M be a symmetric matrix with rows and columns indexed by the elements of S. M is a *binary dissimilarity matrix* on set S if $M(s_i, s_j) \in \{1, 2\}$ for every pair (s_i, s_j) of distinct elements of S and $M(s_i, s_i) = 0$ for every element $s_i \in S$.

A tree T is a 2-*ultrametric* on set S if T is a *rooted* tree whose leaves are labeled by the elements of S and each leaf-to-root path contains exactly two edges. Call a node in T that is neither a leaf nor the root a *middle* node, to avoid ambiguity. The *half-distance* between two leaves s_i and s_j, denoted by $h_T(s_i, s_j)$, is one half of the number of edges on the unique path in T connecting s_i and s_j. Clearly, $h_T(s_i, s_j) \in \{1, 2\}$ if $i \neq j$, and $h_T(s_i, s_i) = 0$ for every i.

Given a binary dissimilarity matrix M and a 2-ultrametric T on set S, define

$$D(T, M) = \sum_{i<j} |h_T(s_i, s_j) - M(s_i, s_j)|,$$

which measures how well T matches the inter-leaf (half-)distances specified by M.[2] The following *fitting ultrametric trees problem* has been shown to be NP-complete by Křivánek and Morávek [7].

FITTING ULTRAMETRIC TREES (FUT). *Given a binary dissimilarity matrix M on set S and a nonnegative integer ℓ, decide if there is a 2-ultrametric T on S such that $D(T, M) \leq \ell$.*

Kearney and Corneil [5] proved that CTRk is NP-hard for any fixed $k \geq 3$ by a (polynomial-time) reduction from FUT (to CTR3). Using a more dexterous reduction, Jiang, Lin, and Xu [4] recently proved that CTR2 is intractable too.

4.1. CPR2. Given a binary dissimilarity matrix M on a set $S = \{s_1, \ldots, s_n\}$, let $S' = \{s_{n+1}, s_{n+2}, \ldots, s_{2n}\}$ be another set of n elements. Define a binary dissimilarity matrix M' on set $S \cup S'$, from M as follows:

For every pair of (not necessarily distinct) integers $i, j \in \{1, 2, \ldots, n\}$,

- $M'(s_i, s_j) = M(s_i, s_j)$;
- $M'(s_{n+i}, s_{n+j}) = M(s_i, s_j)$;
- $M'(s_i, s_{n+j}) = M(s_i, s_j)$ if $i \neq j$;
- $M'(s_i, s_{n+i}) = 1$.

[2] So, here the entries in M are supposed to represent the half-distances between species instead of full distances.

LEMMA 4.1. *If there is a 2-ultrametric T on set S such that $D(T, M) = \ell$, then there is a 2-ultrametric T' on set $S \cup S'$ such that $D(T', M') = 4\ell$.*

Proof. Given a 2-ultrametric T on set S such that $D(T, M) = \ell$, construct a 2-ultrametric T' on set $S \cup S'$ in the following way: The root and middle nodes of T' are the same as those in T; if an $s_i \in S$ is adjacent to a middle node u in T, then both s_i and s_{n+i} are adjacent to u in T'. Clearly, for every pair of (not necessarily distinct) integers $i, j \in \{1, 2, \ldots, n\}$,

- $h_{T'}(s_i, s_j) = h_T(s_i, s_j)$;
- $h_{T'}(s_{n+i}, s_{n+j}) = h_T(s_i, s_j)$;
- $h_{T'}(s_i, s_{n+j}) = h_T(s_i, s_j)$ if $i \neq j$;
- $h_{T'}(s_i, s_{n+i}) = 1$.

Thus, $D(T', M') = 4D(T, M) = 4\ell$. □

LEMMA 4.2. *Let T be a 2-ultrametric on set $S \cup S'$; then there is another 2-ultrametric T' on set $S \cup S'$ such that (1) $D(T', M') \leq D(T, M')$ and (2) for every $i \in \{1, 2, \ldots, n\}$, $s_i \in S$ and $s_{n+i} \in S'$ are adjacent to a common middle node in tree T'.*

Proof. Suppose that $s_i \in S$ is adjacent to a middle node u and $s_{n+i} \in S'$ is adjacent to another middle node $u' \neq u$ in tree T. Let

- $\widetilde{S} = (S \cup S') - \{s_i, s_{n+i}\}$;
- a be the number of $s \in \widetilde{S}$ adjacent to u in T with $M'(s, s_i) = 1$;
- b be the number of $s \in \widetilde{S}$ adjacent to u in T with $M'(s, s_i) = 2$;
- a' be the number of $s \in \widetilde{S}$ adjacent to u' in T with $M'(s, s_{n+i}) = 1$; and
- b' be the number of $s \in \widetilde{S}$ adjacent to u' in T with $M'(s, s_{n+i}) = 2$.

If $a' + b \leq a + b'$, we can modify T by deleting edge (s_{n+i}, u') and adding edge (s_{n+i}, u). Otherwise, we can modify T by deleting edge (s_{n+i}, u) and adding edge (s_{n+i}, u'). In either case, $D(T, M')$ does not increase and, s_i and s_{n+i} are adjacent to a common middle node of the modified T. Repeating this process results in a desired 2-ultrametric. □

From now on, we will only consider those 2-ultrametrics on set $S \cup S'$ such that for every $i \in \{1, \ldots, n\}$, $s_i \in S$ and $s_{n+i} \in S'$ are connected to a common middle node.

LEMMA 4.3. *Given a binary dissimilarity matrix M on set S, there is a 2-ultrametric T on S such that $D(T, M) \leq \ell$ if and only if there is a 2-ultrametric T' on $S \cup S'$ such that $D(T', M') \leq 4\ell$.*

Proof. The "only if" part is implied by Lemmas 4.1 and 4.2. The "if" is straightforward by observing that deleting elements in S' from T' gives a 2-ultrametric T on set S such that $D(T, M) = D(T', M')/4$. □

THEOREM 4.4. CPR2 *is NP-complete.*

Proof. CPR2 is clearly in NP. The hardness proof is done by a reduction from FUT. Consider an instance I of FUT, i.e., a dissimilarity matrix M on set $S = \{s_1, s_2, \ldots, s_n\}$ and a nonnegative integer ℓ. Without loss of generality, we may assume that $n \geq 4$ and $\ell \leq n(n-1)/2$. Construct the corresponding set S' and the dissimilarity matrix M' on $S \cup S'$, from M as in the above. Construct a graph G on a set V of $2n$ vertices as follows. For every $s_i \in S \cup S'$, there is a corresponding vertex v_i in V.[3] Two distinct vertices v_i and v_j are adjacent in G if and only if $M'(s_i, s_j) = 1$.

Let the instance of CPR2 consist of graph G and a nonnegative integer $\ell' = 4\ell$.

[3]We use a different name v_i instead of s_i here in order to avoid ambiguity, although they should be viewed as identical.

We claim that there is an approximate phylogenetic square root T' of graph G with $|E(G) \oplus E(T'^2)| \leq \ell'$ if and only if there is a 2-ultrametric T on set S with $D(T, M) \leq \ell$.

To see the "if" part, suppose that there is a 2-ultrametric T on set S such that $D(T, M) \leq \ell$. This implies there is a 2-ultrametric T'' on set $S \cup S'$ such that $D(T'', M') \leq 4\ell = \ell'$. Recall that for all $i \in \{1, 2, \ldots, n\}$, $s_i \in S$ and $s_{n+i} \in S'$ are adjacent to a common middle node in tree T''. It follows that every middle node in T'' has degree at least 3. Then replacing every leaf s_i in T'' by vertex v_i gives a tree (still denoted by T'') whose leaves are the elements of V. So, if there are three or more middle nodes, then T'' is a phylogeny on V and we are done by setting $T' = T''$. If there is only one middle node in T'', then we obtain T' from T'' by deleting the root as well as its incident edge. If there are exactly two middle nodes in T'', then we obtain T' from T'' by removing the root and connecting the two middle nodes by an edge. Clearly, the final tree T' is a phylogeny on set V. Moreover, $d_{T'}(v_i, v_j) \leq 2$ if and only if $h_{T''}(v_i, v_j) = h_{T''}(s_i, s_j) = 1$ for all distinct $i, j \in \{1, 2, \ldots, n\}$. It follows that edge (v_i, v_j) is in exactly one of $E(G)$ and $E(T'^2)$ if and only if either $h_{T''}(s_i, s_j) = 1$ and $M'(s_i, s_j) = 2$, or $h_{T''}(s_i, s_j) = 2$ and $M'(s_i, s_j) = 1$. That is, the number of such edges is equal to $D(T'', M')$. Thus, $|E(G) \oplus E(T'^2)| = D(T'', M') \leq \ell'$.

To prove the "only if" part, let us assume that T' is a phylogeny interconnecting the vertices in V such that $|E(G) \oplus E(T'^2)| \leq \ell'$. If T' contains only one internal node, i.e., T'^2 is complete, then a 2-ultrametric T'' on set $S \cup S'$ can be constructed to have only one middle node with all elements of $S \cup S'$ attached to it. So, suppose in the following that T' contains two or more internal nodes. We obtain a rooted tree T'' by modifying T' as follows:

1. Select an arbitrary internal edge of T', i.e., an edge connecting two internal nodes, and split the edge into two edges by adding a new internal node, say r, on the edge.
2. Root the new tree at r.
3. Delete all internal edges from the tree. This results in a (possibly empty) set of isolated nodes and a set of stars whose centers are internal nodes of the original T'.
4. Connect the centers of the stars to the root r.

Clearly, the leaves of T'' are the elements of V, T'' is of height 2, and every leaf-to-root path is of length exactly 2. Furthermore, $E(T'^2) = E(T''^2)$. Now replacing leaf v_i in T'' by s_i gives a 2-ultrametric (still denoted by T'') on set $S \cup S'$. Again, an edge (v_i, v_j) is in exactly one of $E(G)$ and $E(T'^2)$ if and only if either $h_{T''}(s_i, s_j) = 1$ and $M'(s_i, s_j) = 2$, or $h_{T''}(s_i, s_j) = 2$ and $M'(s_i, s_j) = 1$. Thus, $D(T'', M') = |E(G) \oplus E(T'^2)| \leq \ell' = 4\ell$. According to Lemmas 4.2 and 4.3, we may easily obtain a 2-ultrametric T on set S, from T'', such that $D(T, M) \leq \ell$. This finishes the proof. □

4.2. CPR*k*. We extend the above NP-completeness result to CPRk for $k > 2$. In doing so, we need to design several gadgets that facilitate the proof.

4.2.1. Gadgets. A *critical clique* [8] of a graph is a maximal subset of vertices that are adjacent to each other and have a common neighborhood. As for constructing a phylogenetic root, the vertices in a critical clique can be identified because they are interchangeable in every phylogenetic root. When we say that one critical clique C_1 is adjacent to another C_2, we mean every vertex in C_1 is adjacent to every vertex in C_2. Consider a graph $H = (V, E)$ consisting of $4k - 7$ critical cliques C_1 through C_{4k-7} such that

- $|C_i| = N$ for $1 \le i \le 4k-7$;
- C_i is adjacent to $C_1, C_2, \ldots, C_{k-2+i}$ for $1 \le i \le k-1$;
- C_i is adjacent to $C_{i-k+2}, C_{i-k+3}, \ldots, C_{i+k-2}$ for $k \le i \le 3k-5$;
- C_i is adjacent to $C_{i-k+2}, C_{i-k+3}, \ldots, C_{4k-7}$ for $3k-4 \le i \le 4k-7$.

Assume that T is an approximate kth root phylogeny of graph H such that $|E(T^k) \oplus E(H)| \le \ell < N$. We present some enforced structural properties of T below.

LEMMA 4.5. *In T, no two vertices from different critical cliques can be adjacent to a common internal node.*

Proof. For the sake of a contradiction, assume that $v_i \in C_i$ and $v_j \in C_j$ with $i \ne j$ are both adjacent to an internal node u in T. Then v_i and v_j have the same neighborhood in the kth phylogenetic power T^k. On the other hand, by the construction of H, there is another critical clique C_h with $h \notin \{i, j\}$ such that C_h is adjacent to exactly one of C_i and C_j in H. So, $|E(T^k) \oplus E(H)|$ is at least as large as N—a contradiction. ∎

Let R be the skeleton obtained from T by deleting all the leaves. In light of Lemma 4.5, a node of R adjacent to some vertex of C_i in tree T is called a *node for* C_i or simply a *C_i-node*. We also call a vertex in clique C_i a *C_i-vertex*. Let R_i be the minimal subtree of R that contains all C_i-nodes. Clearly, R_i can be obtained from R by deleting some nodes and their incident edges. Call R_i the *C_i-subtree*.

LEMMA 4.6. *For every $j \in \{k-1, k, \ldots, 3k-5\}$ and every $i \ne j$, there is no C_i-node in R_j. That is, for every $j \in \{k-1, k, \ldots, 3k-5\}$, the C_j-nodes form a subtree of T.*

Proof. Fix a $j \in \{k-1, k, \ldots, 3k-5\}$. Notice that for every $i \in \{1, 2, \ldots, j-1\}$, if there is a C_i-node, say t_i, in R_j, then t_i must be an internal node in R_j. In tree T, every vertex in C_{j+k-2} is either at distance greater than $k-2$ from some C_j-node or at distance at most $k-2$ from every C_j-node. A C_{j+k-2}-vertex which is at distance at most $k-2$ from every C_j-node is at distance less than $k-2$ from t_i. Therefore, in T^k, each C_{j+k-2}-vertex is either adjacent to no C_j-vertex or adjacent to some C_i-vertex (which is adjacent to t_i in T). This, together with the fact that C_{j+k-2} is adjacent to C_j but adjacent to none of C_1 through C_{j-1}, implies that $|E(T^k) \oplus E(H)| \ge N$—a contradiction. Similarly, using C_{j-k+2} instead of C_{j+k-2}, we can show that R_j contains no C_i-node for every $i \ge j+1$. This proves the lemma. ∎

LEMMA 4.7. *For every $i \in \{k-1, k, \ldots, 3k-5\}$, there is exactly one C_i-node, denoted by t_i, in tree T. Moreover, the nodes $t_{k-1}, t_k, \ldots, t_{3k-5}$ appear consecutively in this order on a path of tree T.*

Proof. Let t_{2k-2} be a C_{2k-2}-node. If t_{2k-2} were within distance $k-2$ from all C_{k-1}-nodes, then the vertices of C_{2k-2} adjacent to t_{2k-2} in tree T would be at distance at most k from every vertex of C_{k-1} in tree T. This would result in at least N edges in $E(T^k) - E(H)$ and is thus impossible. Let t_{k-1} be a C_{k-1}-node such that $d_R(t_{k-1}, t_{2k-2}) > k-2$, and let P denote the path in R connecting them. It holds that for every $i \in \{k, k+1, \ldots, 2k-3\}$, there must be a C_i-node, say t_i, such that $d_R(t_i, t_{k-1}) \le k-2$ and $d_R(t_i, t_{2k-2}) \le k-2$ (otherwise $|E(T^k) \oplus E(H)|$ would be at least N because both C_{k-1} and C_{2k-2} are adjacent to C_i in H). Therefore, in tree R, t_i is closer to some node on path P than to either of t_{k-1} and t_{2k-2}. If subtree R_i contains some node from the path P, then we choose an arbitrary node in $R_i \cap P$ as the *representative* node for R_i. Otherwise, choose the node on P that is the closest to subtree R_i as the representative node for R_i. Denote the representative node for R_i by r_i.

We claim that $r_i \neq r_j$ for every pair of distinct $i, j \in \{k, k+1, \ldots, 2k-3\}$. To prove the claim, for every $i \in \{k, k+1, \ldots, 2k-3\}$, let t'_i be the closest C_i-node to r_i in R. Then $d_R(r_i, t'_i)$ should be less than or equal to both $d_R(r_i, t_{k-1})$ and $d_R(r_i, t_{2k-2})$. It follows that if R_i and R_j with $i < j$ share some representative node, say $r_i = r = r_j$ and $d_R(r, t'_i) \leq d_R(r, t'_j)$ (respectively, $d_R(r, t'_i) \geq d_R(r, t'_j)$), then for every vertex $x \in C_{j+k-2}$ (respectively, $x \in C_{i-k+2}$), either $d_T(x, t'_i) \leq k-1$ or $\max\{d_T(x, t_{2k-2}), d_T(x, t'_j)\} > k-1$ (respectively, either $d_T(x, t'_j) \leq k-1$ or $\max\{d_T(x, t_{k-1}), d_T(x, t'_i)\} > k-1$). This again contradicts the fact that $|E(T^k) \oplus E(H)| < N$.

A similar argument to the proof of the above claim shows that $k \leq i < j \leq 2k-3$ if and only if $d_R(t_{k-1}, r_i) < d_R(t_{k-1}, r_j)$. Since some C_k-node should be within distance $k-2$ from t_{2k-2}, we conclude that the representative node r_k is in fact a C_k-node. Analogously, for every $i \in \{k, k+1, \ldots, 2k-3\}$, r_i is in fact a C_i-node. It further follows that $d_R(t_{k-1}, r_k) = d_R(r_k, r_{k+1}) = \cdots = d_R(r_{2k-3}, t_{2k-2}) = 1$, that is, the path P connecting t_{k-1} and t_{2k-2} is t_{k-1}-r_k-r_{k+1}-…-r_{2k-3}-t_{2k-2}. It is then easy to argue that there is only one C_i-node for every $i \in \{k-1, k, \ldots, 2k-3\}$.

The lemma is proved by considering analogously the C_{2k-4}-node and a C_{3k-5}-node, and the index interval $[2k-4, 3k-5]$. □

By Lemma 4.7, letting t_i denote the unique C_i-node for $i \in \{k-1, k, \ldots, 3k-5\}$, we get a rough structure of R, as shown in Figure 4.1 (where $k = 5$ and t_i indicates a possible C_i-node for $i \in \{1, 2, \ldots, k-2\} \cup \{3k-4, 3k-3, \ldots, 4k-7\}$).

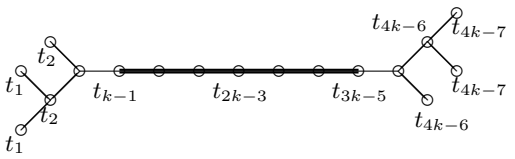


FIG. 4.1. *The rough structure of R.*

4.2.2. Proof of hardness.

THEOREM 4.8. CPRk *is NP-complete when* $k \geq 3$.

Proof. The proof is also a reduction from FUT, but it is more complicated than that of Theorem 4.4. To simplify the presentation, we assume that k is even. It is trivial to extend the proof to odd k. Given a binary dissimilarity matrix M on a set $S = \{s_1, \ldots, s_n\}$, let $S_h = \{s_{hn+1}, s_{hn+2}, \ldots, s_{(h+1)n}\}$ be another set of n elements for all $h = 1, 2, \ldots, 2^{k/2} - 1$. For convenience, let S_0 denote the set S. Define a dissimilarity matrix M' on set $\widetilde{S} = \bigcup_{h=0}^{2^{k/2}-1} S_h$ from M as follows. For every pair of integers $i, j \in \{1, 2, \ldots, n\}$ and every pair of integers $h_1, h_2 \in \{0, 1, \ldots, 2^{k/2} - 1\}$,

- $M'(s_{h_1 n+i}, s_{h_2 n+j}) = M(s_i, s_j)$ if $i \neq j$;
- $M'(s_{h_1 n+i}, s_{h_2 n+i}) = 1$ if $h_1 \neq h_2$;
- $M'(s_{h_1 n+i}, s_{h_1 n+i}) = 0$.

Similarly to the proofs of Lemmas 4.1, 4.2, and 4.3, we can show the following:

(i) There exists a 2-ultrametric T on set S with $D(T, M) \leq \ell$ if and only if there exists a 2-ultrametric T' on set $\widetilde{S}$ with $D(T', M') \leq 2^k \ell$.

(ii) We can consider only those 2-ultrametrics on set $\widetilde{S}$ in which s_{hn+i}, where $h = 0, 1, \ldots, 2^{k/2} - 1$, are adjacent to a common middle node.

Let $\ell' = 2^k\ell$ and $N = \ell' + 1$. The instance of CPRk consists of the nonnegative integer ℓ' and graph G constructed as follows:

- G takes $\widetilde{S}$ as a vertex subset;
- two distinct vertices $s_i, s_j \in \widetilde{S}$ are adjacent in G if and only if $M'(s_i, s_j) = 1$;
- G also contains the graph H consisting of the $4k-7$ critical cliques as discussed in the last subsubsection as a subgraph; and
- for every $i \in \{\frac{3k}{2} - 1, \frac{3k}{2}, \ldots, \frac{5k}{2} - 5\}$, each vertex in C_i is adjacent to every vertex in $\widetilde{S}$.

Lemma 4.7 guarantees that if G has an approximate phylogeny T' such that $|E(G) \oplus E(T'^k)| \le \ell' < N$, then the vertices in each critical clique C_i in subgraph H (where $i \in \{k-1, k, \ldots, 3k-5\}$) are adjacent to the unique internal C_i-node t_i in T'. Since every vertex in $\widetilde{S}$ is adjacent to all vertices in C_i for all $i \in \{\frac{3k}{2} - 1, \frac{3k}{2}, \ldots, \frac{5k}{2} - 5\}$, and the length of the path consisting of those $k-3$ C_i-nodes has length $k-4$, every vertex in $\widetilde{S}$ must be within distance $\frac{k}{2} + 1$ from the (unique) C_{2k-3}-node t_{2k-3} in T'.

Let T'' denote the minimal subtree of T' that contains all vertices in $\widetilde{S}$. The first observation is that T'' contains no vertex outside $\widetilde{S}$. Secondly, since every vertex in $\widetilde{S}$ is adjacent to neither vertex in $C_{\frac{3k}{2}-2}$ nor vertex in $C_{\frac{5k}{2}-4}$, we conclude that every vertex in $\widetilde{S}$ is at distance exactly $\frac{k}{2} + 1$ from the C_{2k-3}-node t_{2k-3}. Furthermore, the path connecting any vertex in $\widetilde{S}$ to t_{2k-3} does not intersect the backbone path formed by the C_i-nodes for $i \in \{k-1, k, \ldots, 3k-5\}$ (except at t_{2k-3}).

Therefore, if subtree T'' does not include node t_{2k-3}, then we can construct a 2-ultrametric tree, denoted also by T'', on set $\widetilde{S}$ by connecting all the elements to a single middle node. Otherwise, rooting T'' at t_{2k-3} and letting every leaf be adjacent to its closest child node of the root, which serves as the middle node, give a 2-ultrametric, denoted still by T'', on set $\widetilde{S}$. In any case, the 2-ultrametric T'' on set $\widetilde{S}$ satisfies $D(T'', M') \le \ell' = 2^k\ell$, which immediately implies that we can construct a 2-ultrametric T on set S such that $D(T, M) \le \ell$.

On the other hand, if we have a 2-ultrametric T on set S such that $D(T, M) \le \ell$, we can easily construct a 2-ultrametric T'' on set $\widetilde{S}$ such that $D(T'', M') \le 2^k\ell = \ell'$ and in which elements s_{hn+i}, $h = 0, 1, \ldots, 2^{k/2} - 1$, are adjacent to a common middle node. It is also easy to transform the subtree of T'' under each middle node into a $\frac{k}{2}$-height subtree rooted at the middle node so that every leaf is at distance exactly $\frac{k}{2}$ from the middle node and every internal node of the whole tree (except its root), still denoted by T'', has degree at least 3. At the same time, we can also easily build a tree for subgraph H in which all vertices in C_i are adjacent to a single C_i-node t_i, and these $4k-7$ C_i-nodes are connected consecutively into a path (such that t_i is adjacent to t_{i-1} and t_{i+1}). We then identify the root of T'' with the C_{2k-3}-node t_{2k-3}. This gives a phylogeny, which is denoted by T', such that $|E(G) \oplus E(T'^k)| = D(T'', M') \le \ell'$. ∎

4.3. Steiner kth root problem. We study another problem closely related to PRk and TRk, which is the *Steiner kth root problem* [8]. Recall that given a graph $G = (V, E)$, TRk asks for a tree whose node set is exactly V, and PRk asks for a tree whose leaf-set is exactly V. A more general problem is to ask for a tree T whose node set is a superset of V and whose leaf-set is a subset of V, and such that for every pair of vertices u and v in V, $d_T(u, v) \le k$ if and only if $(u, v) \in E$. We call a tree T whose node set is a superset of V and whose leaf-set is a subset of V a *Steiner tree* on V.

STEINER kTH ROOT PROBLEM (SRk). *Given a graph $G = (V, E)$, find a Steiner tree T on V such that for each pair of vertices $u, v \in V$, $(u, v) \in E$ if and only if $d_T(u, v) \le k$.*

Such a Steiner tree T (if it exists) is called a *Steiner kth root* or a *kth root Steiner tree* of G. G is called the *kth Steiner power* of T. We also abuse T^k to denote the *kth Steiner power* of T when there is no confusion from the context.

Notice that here we do not require a nonleaf node in a Steiner tree to have degree at least 3. This requirement is not necessary from the tree root point of view. But one may do so, as this requirement is natural from the phylogenetic root point of view. Steiner trees satisfying this additional requirement are called *restricted Steiner trees*. Graphs having a restricted Steiner kth root for $k = 1, 2$ can be recognized in linear time [8]. The recognition algorithm can be extended to find an ordinary Steiner kth root for $k = 1$ and $k = 2$. However, when $k \geq 3$, no polynomial-time recognition algorithm has been reported yet to find either a Steiner kth root or a restricted Steiner kth root. In the following, we will consider only ordinary Steiner roots and show that the closest Steiner kth root problem (CSRk), defined in a straightforward way, is NP-complete when $k \geq 2$.

We call the nodes in a Steiner tree T that are not vertices in V *Steiner nodes*.

For CSR1, we notice that deleting all Steiner nodes from an (approximate) 1st root Steiner tree T results in a collection of subtrees such that vertices in different subtrees are not adjacent in T^1. Therefore, for any input graph G, the best way to build the closest 1st root Steiner tree is to construct a spanning tree for each connected component in G and then connect these spanning trees together via a Steiner node. That is, a closest 1st root Steiner tree can be computed in $O(n)$ time, where n is the number of vertices in the input graph. The complexity changes when k marches from 1 to 2. In fact, by designing similar yet more careful gadgets, we are able to construct a reduction from FUT to CSRk for every fixed $k \geq 2$.

THEOREM 4.9. CSRk *is NP-complete for every* $k \geq 2$.

We omit the proof of Theorem 4.9 in this paper because it is quite long and somewhat tedious (especially given the proof of Theorem 4.8). The complete proof can be found in the technical report [3].

5. Open problems. Since CPRk is NP-complete for all $k \geq 2$, it would be interesting to know how well we can approximate the closest phylogenetic kth root. Also, it would be nice to extend Theorem 3.7 and Corollary 3.8 to disconnected graphs.

Acknowledgments. The authors thank the referee for helpful comments. The third author thanks Paul Kearney for bringing CPRk to his attention and thanks Bin Ma and Jinbo Xu for many helpful discussions.

REFERENCES

[1] H. L. BODLAENDER, *NC-algorithms for graphs with small treewidth*, in Proceedings of the 14th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1988), Lecture Notes in Comput. Sci. 344, Springer-Verlag, Berlin, 1989, pp. 1–10.

[2] A. BRANDSTÄDT, V. B. LE, AND J. P. SPINRAD, *Graph Classes: A Survey*, SIAM Monogr. Discrete Math. Appl. 3, SIAM, Philadelphia, 1999.

[3] Z.-Z. CHEN, T. JIANG, AND G.-H. LIN, *Computing Phylogenetic Roots with Bounded Degrees and Errors*, Technical Report TR03-06, Department of Computing Science, University of Alberta, 2003.

[4] T. JIANG, G.-H. LIN, AND J. XU, *On the Closest Tree kth Root Problem*, manuscript, Department of Computer Science, University of Waterloo, 2000.

[5] P. E. KEARNEY AND D. G. CORNEIL, *Tree powers*, J. Algorithms, 29 (1998), pp. 111–131.

[6] T. KLOKS, *Treewidth*, Lecture Notes in Comput. Sci. 842, Springer-Verlag, Berlin, 1994.

[7] M. KŘIVÁNEK AND J. MORÁREK, *NP-hard problems in hierarchical-tree clustering*, Acta Inform., 23 (1986), pp. 311–323.

[8] G.-H. Lin, P. E. Kearney, and T. Jiang, *Phylogenetic k-root and Steiner k-root*, in Proceedings of the 11th Annual International Symposium on Algorithms and Computation (ISAAC 2000), Lecture Notes in Comput. Sci. 1969, Springer-Verlag, Berlin, 2000, pp. 539–551.

[9] Y.-L. Lin and S. S. Skiena, *Algorithms for square roots of graphs*, SIAM J. Discrete Math., 8 (1995), pp. 99–118.

[10] R. Motwani and M. Sudan, *Computing roots of graphs is hard*, Discrete Appl. Math., 54 (1994), pp. 81–88.

[11] N. Nishimura, P. Ragde, and D. M. Thilikos, *On graph powers for leaf-labeled trees*, in Proceedings of the 7th Scandinavian Workshop on Algorithm Theory (SWAT 2000), Lecture Notes in Comput. Sci. 1851, Springer-Verlag, Berlin, 2000, pp. 125–138.

[12] D. L. Swofford, G. J. Olsen, P. J. Waddell, and D. M. Hillis, *Phylogenetic inference*, in Molecular Systematics, 2nd ed., D. M. Hillis, C. Moritz, and B. K. Mable, eds., Sinauer Associates, Sunderland, MA, 1996, pp. 407–514.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 880–903

LEXICOGRAPHICAL GENERATION OF A GENERALIZED DYCK LANGUAGE*

JENS LIEBEHENSCHEL†

Abstract. We consider a generalization of the Dyck language. We have a set of opening and a set of closing brackets and a relation that gives the pairs of brackets that can be used in the generalized Dyck language. This Dyck language is a handy coding for several classes of labeled trees.

We present an algorithm that generates all words of length $2n$ of the generalized Dyck language lexicographically. Thereby, each word is computed from its predecessor according to the lexicographical order without any knowledge about the words generated before.

Additionally, we introduce a condition on the relation for the generalized Dyck language to be simply generated, which means that an algorithm needs to read only the suffix to be changed in order to compute the successor of a word according to the lexicographical order. We present an algorithm that checks whether a Dyck language is simply generated or not.

For an arbitrary relation, we compute the sth moments of the random variable describing the length of the suffix to be changed in the computation of the successor of a Dyck word. In particular, the mean value and the variance are constant.

Key words. Dyck language, lexicographical generation, analysis of algorithms

AMS subject classifications. 05A15, 05C05, 68R05, 68W40

DOI. 10.1137/S0097539701394493

1. Overview and definitions.

In this section, we introduce the generalization of the Dyck language, the lexicographical order needed for the lexicographical generation, and all definitions—illustrated by several examples—for the whole paper. We also point out the contents of the following sections.

In this paper, we present an algorithm that generates all words of length $2n$ of the generalized Dyck language given in Definition 1.1 lexicographically. The algorithm reads a word from right to left and changes a suffix of that word in order to generate the next word according to the lexicographical order given in Definition 1.2.

DEFINITION 1.1. *Let* $T_{\lfloor} := \{\lfloor_1, \lfloor_2, \ldots, \lfloor_{t_1}\}$ *(resp.,* $T_{\rfloor} := \{\rfloor_1, \rfloor_2, \ldots, \rfloor_{t_2}\}$*) with* t_1, $t_2 \in \mathbb{N}$ *be the* set of opening *(resp.,* closing*)* brackets*. Let* $|S|$ *be the cardinality of the set* S*, so* $|T_{\lfloor}| = t_1$ *and* $|T_{\rfloor}| = t_2$*. With* $T := T_{\lfloor} \dot{\cup} T_{\rfloor}$*, where* $\dot{\cup}$ *denotes the* disjoint union *of sets, and a relation* $\mathcal{R} \subseteq T_{\lfloor} \times T_{\rfloor}$*, we obtain the* generalized Dyck language associated with $\mathcal{R}$ *(cf.* [Ke96]*) by*

$$D := \{w \in T^\star \mid w \equiv \varepsilon \bmod \delta\},$$

where ε *denotes the* empty word *and* δ *is the congruence over* T *which is defined by* $(\forall(\lfloor_a, \rfloor_b) \in \mathcal{R})(\lfloor_a \rfloor_b \equiv \varepsilon \bmod \delta)$*. The* set of all Dyck words of length $2n$ *is given by* $D_{2n} := D \cap T^{2n}$.

Remark 1.1. Obviously, there is a unique *corresponding closing* (resp., *opening*) bracket to each opening (resp., closing) bracket in every Dyck word $w \in D$.

*Received by the editors August 28, 2001; accepted for publication (in revised form) September 8, 2002; published electronically June 10, 2003.
http://www.siam.org/journals/sicomp/32-4/39449.html
†Fachbereich Biologie und Informatik, Institut für Informatik, Johann Wolfgang Goethe-Universität, Frankfurt am Main, D-60054 Frankfurt am Main, Germany (jens@sads.informatik.uni-frankfurt.de).

The closing bracket corresponding to an opening bracket in a Dyck word $w \in D$ can be found by searching for the first closing bracket behind the shortest word $w_2 \in D$ (w_2 might be ε) on the right side of the opening bracket. If the opening (resp., its corresponding closing) bracket is $[_a$ (resp., $]_b$), we have $w = w_1\,[_a\,w_2\,]_b\,w_3$ with $w_1\,w_3, w_2 \in D$. The corresponding opening bracket to a closing bracket can be found in an analogous way.

DEFINITION 1.2. *Let $\lessdot \subseteq T \times T$ be an irreflexive linear ordering on T. The* lexicographical order $\prec$ over T^+ *is defined as the extension of $\lessdot$ to $\prec \subseteq T^+ \times T^+$ by*

$$\begin{aligned} x \prec y :\Longleftrightarrow \quad & \big(\exists z \in T^+\big)(xz = y) \\ & \vee \big(\exists\,(w, x', y', a, b) \in T^{\star 3} \times T^2\big)(x = wax' \wedge y = wby' \wedge a \lessdot b)\,, \end{aligned}$$

cf. [Ke98]. *Note that in this paper we consider the lexicographical order on words of length $2n$ only.*

We use the following ordering on T:

$$[_{|T_[|} \lessdot \cdots \lessdot [_1 \lessdot\ ]_1 \lessdot \cdots \lessdot\ ]_{|T_]|}.$$

Now, let us have a closer look at the relation $\mathcal{R}$. From the existence of the lexicographical order $\prec$ on D results the existence of a unique *lexicographically minimal (resp., maximal) tuple in* $\mathcal{R}$ which is denoted by $\mathcal{R}_{\min}$ (resp., $\mathcal{R}_{\max}$).

DEFINITION 1.3. *Let $p_{\min}$ (resp., $p_{\max}$) be the* lexicographically minimal *(resp., maximal) pair of brackets in $\mathcal{R}$. Further, let $[_{p_{\min}}, [_{p_{\max}} \in T_[$ and $\ ]_{p_{\min}},]_{p_{\max}} \in T_]$. Then we get*

$$\begin{aligned} p_{\min} := [_{p_{\min}}\,]_{p_{\min}} &\iff \mathcal{R}_{\min} = \big([_{p_{\min}},]_{p_{\min}}\big) \text{ and} \\ p_{\max} := [_{p_{\max}}\,]_{p_{\max}} &\iff \mathcal{R}_{\max} = \big([_{p_{\max}},]_{p_{\max}}\big). \end{aligned}$$

Obviously, $[_{p_{\min}}$ (resp., $]_{p_{\min}}$) is the opening *(resp.,* closing*)* bracket of the lexicographically minimal pair of brackets, *and $[_{p_{\max}}$ (resp., $]_{p_{\max}}$) is the* opening *(resp.,* closing*)* bracket of the lexicographically maximal pair of brackets. *Now, we are able to compute the* minimal *(resp.,* maximal*)* word of D_{2n} according to the lexicographical order $w_{\min}^{D_{2n}}$ *(resp., $w_{\max}^{D_{2n}}$)*:

$$w_{\min}^{D_{2n}} = \big([_{p_{\min}}\big)^n \big(]_{p_{\min}}\big)^n \quad \text{and} \quad w_{\max}^{D_{2n}} = \big(p_{\max}\big)^n.$$

In this paper, we assume that there are no useless symbols in T, so we find $[_{p_{\min}} = [_{|T_[|}$ and $[_{p_{\max}} = [_1$.

DEFINITION 1.4. *Given $([_a,]_b) \in \mathcal{R}$, $]_b$ is called* minimal *(resp.,* maximal*)* with respect to $[_a$ *if there is no smaller (resp., larger) closing bracket that corresponds to $[_a$ in $\mathcal{R}$. For the sake of clear presentation, we use "minimal (resp., maximal)" instead of "minimal (resp., maximal) with respect to the corresponding opening bracket". A string of closing brackets is called minimal (resp., maximal) if each bracket in the string is minimal (resp., maximal).*

Note that the property of being minimal (resp., maximal) depends not only on the closing bracket, but also on the corresponding opening bracket. For example, $]_b$ of $([_{a_1},]_b)$ can be minimal even if $]_b$ of $([_{a_2},]_b)$ is not minimal.

Let us demonstrate the above definitions by two simple examples.

Example 1.1. Let $T := \{[_1, [_2,]_1,]_2\}$, $\mathcal{R} := \{([_1,]_1), ([_2,]_2)\}$, and $n := 2$. We get the ordering on the alphabet $[_2 \lessdot [_1 \lessdot\ ]_1 \lessdot\ ]_2$. The lexicographically minimal (resp.,

$$\begin{array}{ccccccccccccc} & & & & [_2\,[_2\,]_2\,]_2 & \prec & [_2\,[_1\,]_1\,]_2 & \prec & [_2\,]_2\,[_2\,]_2 & \prec & [_2\,]_2\,[_1\,]_1 \\ \prec & [_1\,[_2\,]_2\,]_1 & \prec & [_1\,[_1\,]_1\,]_1 & \prec & [_1\,]_1\,[_2\,]_2 & \prec & [_1\,]_1\,[_1\,]_1 & & & \end{array}$$

FIG. 1. *All Dyck words of Example* 1.1 *arranged according to the lexicographical order* $\prec$.

$$\begin{array}{ccccccccccccc} & & [_2\,[_2\,]_2\,]_2 & \prec & [_2\,[_1\,]_1\,]_2 & \prec & [_2\,[_1\,]_2\,]_2 & \prec & [_2\,]_2\,[_2\,]_2 & \prec & [_2\,]_2\,[_1\,]_1 & \prec & [_2\,]_2\,[_1\,]_2 \\ \prec & [_1\,[_2\,]_2\,]_1 & \prec & [_1\,[_2\,]_2\,]_2 & \prec & [_1\,[_1\,]_1\,]_1 & \prec & [_1\,[_1\,]_1\,]_2 & \prec & [_1\,[_1\,]_2\,]_1 & \prec & [_1\,[_1\,]_2\,]_2 & \\ \prec & [_1\,]_1\,[_2\,]_2 & \prec & [_1\,]_1\,[_1\,]_1 & \prec & [_1\,]_1\,[_1\,]_2 & \prec & [_1\,]_2\,[_2\,]_2 & \prec & [_1\,]_2\,[_1\,]_1 & \prec & [_1\,]_2\,[_1\,]_2 & \end{array}$$

FIG. 2. *All Dyck words of Example* 1.2 *arranged according to the lexicographical order* $\prec$.

maximal) word is given by $w_{\min}^{D_4} = [_2\,[_2\,]_2\,]_2$ (resp., $w_{\max}^{D_4} = [_1\,]_1\,[_1\,]_1$). In Figure 1, we find all Dyck words of this example.

Example 1.2. Let $T := \{[_1, [_2,]_1,]_2\}$, $\mathcal{R} := \{([_1,]_1), ([_1,]_2), ([_2,]_2)\}$, and $n := 2$. Again, we obtain the ordering on the alphabet $[_2 \lessdot [_1 \lessdot\,]_1 \lessdot\,]_2$. The lexicographically minimal (resp., maximal) word is given by $w_{\min}^{D_4} = [_2\,[_2\,]_2\,]_2$ (resp., $w_{\max}^{D_4} = [_1\,]_2\,[_1\,]_2$). Given $([_1,]_1)$, we see that $]_1$ is minimal but not maximal. Analogously, $]_2$ of $([_1,]_2)$ is maximal but not minimal. Regarding $([_2,]_2)$, we find $]_2$ to be both minimal and maximal. In Figure 2, all Dyck words of this example are arranged lexicographically.

Now, in the following two definitions, let us recall some functions defined in [Ke98]; the formal definitions can be found in [Li98] as well.

DEFINITION 1.5. *For an arbitrary language* L *and* $w \in L$, *the* successor function $\mathsf{next}(w)$ *computes the* successor of w according to the lexicographical order; $\mathsf{next}(w_{\max}^L)$ *is undefined.*

DEFINITION 1.6. *Let* $\mathsf{pre}(w)$ *be the* longest common prefix of w and $\mathsf{next}(w)$; $\mathsf{old}(w)$ (*resp.,* $\mathsf{new}(w)$) *is the* suffix of w (*resp.,* $\mathsf{next}(w)$) to the right of $\mathsf{pre}(w)$. *Thus* $w = \mathsf{pre}(w)\mathsf{old}(w)$ *and* $\mathsf{next}(w) = \mathsf{pre}(w)\mathsf{new}(w)$ *are factorizations of* w *and its successor. The language* $L \subseteq \Sigma^\star$ *is called* simply generated *with respect to* $\prec$ *iff it satisfies the property*

$$\big(\forall (w, w') \in L^2\big)\big(\mathsf{old}(w) = \vartheta\,\mathsf{old}(w'), \vartheta \in \Sigma^\star \rightsquigarrow \big(\vartheta, \mathsf{new}(w)\big) = \big(\varepsilon, \mathsf{new}(w')\big)\big).$$

This means that a language L is called simply generated if it is possible to determine the suffix $\mathsf{old}(w)$ and replace it by the suffix $\mathsf{new}(w)$ in a unique way for all $w \in L \setminus \{w_{\max}^L\}$ without any knowledge about $\mathsf{pre}(w)$. So, a language L is called simply generated if it is sufficient to read the suffix to be changed only for all $w \in L \setminus \{w_{\max}^L\}$ in order to compute $\mathsf{next}(w)$.

Example 1.3. Considering Example 1.1, with $w = [_2\,[_2\,]_2\,]_2$ we find $\mathsf{next}(w) = [_2\,[_1\,]_1\,]_2$. Obviously, we get $\mathsf{pre}(w) = [_2$, $\mathsf{old}(w) = [_2\,]_2\,]_2$, and $\mathsf{new}(w) = [_1\,]_1\,]_2$. It is easy to check that the Dyck language D_4 with $\mathcal{R} := \{([_1,]_1), ([_2,]_2)\}$ is simply generated.

In section 2, we formalize the successor function; for that purpose we have to define some functions that give information on the relation $\mathcal{R}$. In the algorithm, we use the notation $w = w_0 \dots w_{2n-1} \in D_{2n}$ with $w_i \in T$, $0 \le i \le 2n-1$.

The first three functions depend on the relation $\mathcal{R}$ and a given Dyck word w. Their formal definitions can be found in [Li98].

The function $\mathsf{succ\_pair}$. The function $\mathsf{succ\_pair}([_a\,]_b)$ computes to a given pair of brackets $[_a\,]_b$ the next pair of brackets according to the lexicographical order $\prec$ and

the relation $\mathcal{R}$. Thus it computes the successor of a Dyck word of length two.

Note that for all $(r_1, r_2) \in \mathcal{R} \setminus \{\mathcal{R}_{\max}\}$, the function $\mathsf{succ\_pair}(r_1\, r_2)$ is always defined.

The function succ. The function $\mathsf{succ}(w, i)$ computes to a given closing bracket $w_i =]_b$ in a Dyck word the next closing bracket according to the lexicographical order $\prec$ and the relation $\mathcal{R}$. For some closing brackets, the unique corresponding opening bracket to $]_b$ in the Dyck word w has to be known. It can be determined as described in Remark 1.1.

Note that, for all closing brackets that are not maximal, succ is always defined.

The function min_pred. Given a Dyck word with $w_i =]_b$, and $[_a$ being the corresponding opening bracket to $]_b$, the function $\mathsf{min\_pred}(w, i)$ computes the minimal closing bracket that corresponds to $[_a$ according to the lexicographical order $\prec$ and the relation $\mathcal{R}$.

Note that, for all closing brackets, min_pred is always defined. Further, the function min_pred can be applied to $w_i \dots w_j \in T_]^\star$; we define

$$\mathsf{min\_pred}(w, i, j) := \mathsf{min\_pred}(w, i)\, \mathsf{min\_pred}(w, i+1) \dots \mathsf{min\_pred}(w, j).$$

If $w_i \dots w_j = \varepsilon$, we obtain $\mathsf{min\_pred}(w, i, j) = \varepsilon$.

Remark 1.2. The functions succ and min_pred need some more information, namely, the corresponding opening bracket to the function's argument, which is a closing bracket. For some relations $\mathcal{R}$ it is necessary to read that opening bracket; for others it is not necessary, because the information needed can be determined by the relation $\mathcal{R}$. A condition for this requirement of information will be given in section 3.

The following boolean functions depend on the relation $\mathcal{R}$ only.

The function no_succ. The function $\mathsf{no\_succ}(\,]_b)$ is true iff the given closing bracket $]_b$ is maximal with respect to all pairs of brackets in which it appears.

The function succ_unique. The function $\mathsf{succ\_unique}(\,]_b)$ is true iff the call of the function succ for any closing bracket $]_b$ leads to the same result—independently of the corresponding opening bracket to $]_b$.

The function min_pred_unique. The function $\mathsf{min\_pred\_unique}(\,]_b)$ is true iff the result of min_pred for a given closing bracket $]_b$ is the same—independently of the corresponding opening bracket to $]_b$.

Again, these three functions are formally defined in [Li98].

Example 1.4. Let us revisit Example 1.2 with $\mathcal{R} := \{([_1,]_1), ([_1,]_2), ([_2,]_2)\}$. As the lexicographical order on the pairs of brackets is given by $[_2\,]_2 \prec [_1\,]_1 \prec [_1\,]_2$, we obtain $\mathsf{succ\_pair}([_2\,]_2) = [_1\,]_1$, $\mathsf{succ\_pair}([_1\,]_1) = [_1\,]_2$, $\mathsf{succ\_pair}([_1\,]_2) =$ undefined. Now, we regard

$$w = w_0\, w_1\, w_2\, w_3\, w_4\, w_5 = [_1\,]_2\, [_1\, [_2\,]_2\,]_1 \in D_6.$$

Obviously,

$$\mathsf{succ}(w, 1) = \text{undefined}, \mathsf{succ}(w, 4) = \text{undefined}, \mathsf{succ}(w, 5) =]_2,$$

$$\mathsf{min\_pred}(w, 1) =]_1, \mathsf{min\_pred}(w, 4, 5) =]_2\,]_1.$$

The boolean functions immediately yield

$$\mathsf{no\_succ}(\,]_1) = \text{false}, \mathsf{no\_succ}(\,]_2) = \text{true},$$

$$\mathsf{succ\_unique}(\,]_1) = \text{true}, \mathsf{succ\_unique}(\,]_2) = \text{true},$$

$$\mathsf{min\_pred\_unique}(\,]_1) = \text{true}, \mathsf{min\_pred\_unique}(\,]_2) = \text{false}.$$

In section 3, we formalize the condition on the relation $\mathcal{R}$ whether a given language D_{2n} is simply generated or not. Further, we present an algorithm that decides whether or not a relation $\mathcal{R}$ results in a simply generated Dyck language. The following two definitions are needed for section 3.

DEFINITION 1.7. *Let* $\mathsf{open}(\,]_j) := \{[_i \mid ([_i,]_j) \in \mathcal{R}\}$ *(resp.,* $\mathsf{close}([_i) := \{\,]_j \mid ([_i,]_j) \in \mathcal{R}\}$*) be the* set of all corresponding opening (resp., closing) brackets to a given closing (resp., opening) bracket according to the relation $\mathcal{R}$.

Further, let $\mathsf{successor}(\,]_j, [_i)$ *be the function that computes the same closing bracket for a relation* $\mathcal{R}$ *as* succ *does for a closing bracket of a Dyck word. The only difference is that the function needs to know the corresponding closing bracket to the opening bracket, because it is not able to determine it as* succ*, because* succ *gets the whole word as a parameter and not only a closing bracket.*

For the formal definition of successor*, refer to* [Li98].

Note that open, close, and successor depend on the relation $\mathcal{R}$ only and not on a Dyck word.

DEFINITION 1.8. *The* matrix representation *of a relation* $\mathcal{R}$ *is given by*

$$M := (m_{i,j})_{1\le i\le|T_]|,1\le j\le|T_]|} \quad \textit{with} \quad m_{i,j} := \begin{cases} 1 & \textit{if} \quad ([_i,]_j) \in \mathcal{R}, \\ 0 & \textit{if} \quad ([_i,]_j) \notin \mathcal{R}. \end{cases}$$

Example 1.5. Considering the relation $\mathcal{R} := \{([_1,]_1), ([_1,]_2), ([_2,]_2)\}$ of Example 1.2, we immediately find $\mathsf{open}(\,]_1) = \{[_1\}$, $\mathsf{open}(\,]_2) = \{[_1, [_2\}$, $\mathsf{close}([_1) = \{\,]_1,]_2\}$, and $\mathsf{close}([_2) = \{\,]_2\}$. The matrix representation of $\mathcal{R}$ is given by $M = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$.

In section 4, we analyze the length of the suffix to be changed in order to generate the next word according to the lexicographical order on the average. This average-case analysis is based on a general approach to the average length of the shortest suffix to be changed when generating words of a language lexicographically, cf. [Ke98].

In section 5, we point out that the generalized Dyck language is a handy coding for several classes of labeled trees. Hence we are able to generate classes of labeled trees lexicographically.

2. Algorithm for the generation of Dyck words. In this section, we will present an algorithm that generates all words $w \in D_{2n}$ lexicographically. We follow a classical method for lexicographical generation of combinatorial objects, cf. [Wi77], [NW78], and [Ke98]; the lexicographical generation will be carried out by successive calls of a function which obtains a word and computes its successor according to the lexicographical order. The successor is computed by changing a suffix of the word.

In contrast to the above method, all objects of a desired length can be generated modifying all (recursively generated) objects of the next shorter length, cf. [BLP98]. The latter method does not allow the computation of the successor of a word, if

```
function generate ()
  begin
    w := w_min^{D_2n}
      while w ≠ undefined do
        w := next(w)
  end;
```

FIG. 3. *Algorithm* 1. *Function that generates all words of the language D_{2n} lexicographically.*

merely that word is known; at least one word of shorter length is required. Thus, for saving the state of the generation process, not less than one word of each length has to be stored. In the method used here, only one word needs to be saved. So, we can interrupt the computation without extra cost.

The reason for generating according to the lexicographical order is the following: For the lexicographical generation (following the classical method), it is known that the mean length of the suffix to be changed is as short as possible, cf. [Ke98]. Our algorithm is optimal with respect to the length of the suffix to be changed.

The function generate (see Algorithm 1) starts with the generation of the lexicographically minimal word $w_{\min}^{D_{2n}}$ and successively generates the successor $\mathsf{next}(w)$ of $w \in D_{2n}$ according to the lexicographical order until the lexicographically maximal word $w_{\max}^{D_{2n}}$ has been generated. Remember that $\mathsf{next}(w_{\max}^{D_{2n}})$ is undefined.

Note that in the function generate, the lexicographical generation is a transformation from one word to its successor. So, each word $w \in D_{2n} \setminus \{w_{\min}^{D_{2n}}\}$ depends on its *predecessor* only; there is no need for more information on the words previously generated.

For the function next we need two technical lemmas. In Lemma 2.1, we prove that D_{2n} can be split into four disjoint sets. In Lemma 2.2, we show that each Dyck word $w = w_0 \dots w_{2n-1} \in D_{2n}$ has a unique factorization. The existence of the unique factorization is required for the function next.

LEMMA 2.1. *Let $S_{2n}^{(j)} \subseteq D_{2n}$, $1 \le j \le 4$. With $w \in D_{2n}$, the following hold:*

$$w \in S_{2n}^{(1)} \iff w = w_{\max}^{D_{2n}},$$

$$w \in S_{2n}^{(2)} \iff w = \underbrace{x}_{\in T^+} \quad \underbrace{w_\alpha}_{\substack{\in T_\rfloor \\ \text{not maximal}}} \quad \underbrace{w_{\alpha+1} \dots w_\beta}_{\substack{\in T_\rfloor^\star \\ \text{maximal}}} \quad \underbrace{w_{\max}^{D_{2i}}}_{\substack{\text{with} \\ 0 \le i \le n-1}},$$

$$w \in S_{2n}^{(3)} \iff w = \underbrace{x}_{\in T^\star} \quad \underbrace{w_\alpha\, w_{\alpha+1}}_{\substack{\in D_2 \setminus \{w_{\max}^{D_2}\} \\ w_{\alpha+1} \text{ maximal}}} \quad \underbrace{w_{\alpha+2} \dots w_\beta}_{\substack{\in T_\rfloor^\star \\ \text{maximal}}} \quad \underbrace{w_{\max}^{D_{2i}}}_{\substack{\text{with} \\ 0 \le i \le n-1}},$$

$$w \in S_{2n}^{(4)} \iff w = \underbrace{x}_{\in T^+} \quad w_{\max}^{D_2} \quad \underbrace{w_\alpha \dots w_\beta}_{\substack{\in T_\rfloor^+ \\ \text{maximal}}} \quad \underbrace{w_{\max}^{D_{2i}}}_{\substack{\text{with} \\ 0 \le i \le n-2}}.$$

Then the following equation holds:

$$D_{2n} = S_{2n}^{(1)} \,\dot\cup\, S_{2n}^{(2)} \,\dot\cup\, S_{2n}^{(3)} \,\dot\cup\, S_{2n}^{(4)}.$$

Proof. We present only the sketch of the proof here; the proof is worked out in [Li98] in detail.

With $S_{2n}^{(j)} \subseteq D_{2n}$, $1 \le j \le 4$, it follows that every word that is not a Dyck word is in none of these sets.

Now, we have to prove that every Dyck word is in at least one set. We assume that a Dyck word w exists which is in none of the sets $S_{2n}^{(j)}$, $1 \le j \le 4$. Then we factorize w by splitting off first the longest suffix $p_{\max} \dots p_{\max}$ and second the longest suffix consisting of maximal closing brackets only. Several cases arise, and in each case we obtain a contradiction; thus each Dyck word is in at least one of the sets defined above.

Finally, we just have to prove that the sets $S_{2n}^{(1)}$, $S_{2n}^{(2)}$, $S_{2n}^{(3)}$, and $S_{2n}^{(4)}$ are pairwise disjoint. We obtain six cases; in each case, the assumption that a Dyck word is in both sets leads to a contradiction. □

Remark 2.1. If $n = 1$, we find $S_2^{(4)} = \emptyset$.

LEMMA 2.2. *Each Dyck word $w \in D_{2n}$ has a unique factorization.*

Proof. Again, we outline the proof; the formal proof can be found in [Li98].

As we have already shown that $S_{2n}^{(1)}$, $S_{2n}^{(2)}$, $S_{2n}^{(3)}$, and $S_{2n}^{(4)}$ are pairwise disjoint, we have only to prove that, for each word w, there is only one factorization according to the set it belongs to. The assumption to find two different factorizations for a Dyck word leads to a contradiction for each set. □

Now we are able to point out the successor function $\mathsf{next}(w)$ of a word $w \in D_{2n}$ according to the lexicographical order. With Lemmas 2.1 and 2.2, we can compute the successor for each of the sets $S_{2n}^{(j)}$, $1 \le j \le 4$, separately.

THEOREM 2.3. *The successor of a Dyck word $w \in D_{2n}$ is computed as follows.*

$w \in S_{2n}^{(1)}$:

$$\begin{aligned} w &= w_{\max}^{D_{2n}} \\ \mathsf{next}(w) &= \textit{undefined} \end{aligned}$$

$w \in S_{2n}^{(2)}$:

$$\begin{array}{rcl} w & = & \overbrace{x}^{\in T^+}\ \overbrace{w_\alpha}^{\in T_\rfloor \textit{ not maximal}}\ \overbrace{w_{\alpha+1} \dots w_\beta}^{\in T_\rfloor^\star \textit{ maximal}}\ \overbrace{w_{\max}^{D_{2i}}}^{\textit{with } 0 \le i \le n-1} \\ \mathsf{next}(w) & = & \underbrace{x}_{\in T^+}\ \underbrace{\mathsf{succ}(w,\alpha)}_{\in T_\rfloor}\ \underbrace{w_{\min}^{D_{2i}}}_{\substack{\textit{with} \\ 0 \le i \le n-1}}\ \underbrace{\mathsf{min\_pred}(w,\alpha+1,\beta)}_{\substack{\in T_\rfloor^\star \\ \textit{minimal}}} \end{array}$$

$w \in S_{2n}^{(3)}$:

$$\begin{array}{rcl} w & = & \overbrace{x}^{\in T^\star}\ \overbrace{w_\alpha\, w_{\alpha+1}}^{\substack{\in D_2 \setminus \{w_{\max}^{D_2}\} \\ w_{\alpha+1}\ \textit{maximal}}}\ \overbrace{w_{\alpha+2} \dots w_\beta}^{\in T_\rfloor^\star \textit{ maximal}}\ \overbrace{w_{\max}^{D_{2i}}}^{\textit{with } 0 \le i \le n-1} \\ \mathsf{next}(w) & = & \underbrace{x}_{\in T^\star}\ \underbrace{\widehat{w_\alpha}}_{\in T_\lfloor}\ \underbrace{w_{\min}^{D_{2i}}}_{\substack{\textit{with} \\ 0 \le i \le n-1}}\ \underbrace{\widehat{w_{\alpha+1}}}_{\substack{\in T_\rfloor \\ \textit{minimal}}}\ \underbrace{\mathsf{min\_pred}(w,\alpha+2,\beta)}_{\substack{\in T_\rfloor^\star \\ \textit{minimal}}} \end{array}.$$

Here, $\mathsf{succ\_pair}(w_\alpha\, w_{\alpha+1}) = \widehat{w_\alpha}\ \widehat{w_{\alpha+1}}$.

$$
\begin{array}{lcl}
w \in S_{2n}^{(4)}: & & \\
w & = & \overbrace{x}^{\in T^+}\ w_{\max}^{D_2}\ \overbrace{w_\alpha}^{\substack{\in T_\rfloor \\ maximal}}\ \overbrace{w_{\alpha+1}\ldots w_\beta}^{\substack{\in T_\rfloor^\star \\ maximal}}\ \overbrace{w_{\max}^{D_{2i}}}^{\substack{with \\ 0\le i\le n-2}} \\
\mathsf{next}(w) & = & \underbrace{x}_{\in T^+}\ \underbrace{\mathsf{min\_pred}(w,\alpha)}_{\substack{\in T_\rfloor \\ minimal}}\ \underbrace{w_{\min}^{D_{2(i+1)}}}_{\substack{with \\ 0\le i\le n-2}}\ \underbrace{\mathsf{min\_pred}(w,\alpha+1,\beta)}_{\substack{\in T_\rfloor^\star \\ minimal}}\ .
\end{array}
$$

Proof. As the proofs for the sets $S_{2n}^{(2)}$, $S_{2n}^{(3)}$, and $S_{2n}^{(4)}$ are similar, some cases can be omitted here. All cases are treated in [Li98] completely.

$w \in S_{2n}^{(1)}$: $\rightsquigarrow w = w_{\max}^{D_{2n}}$; $w_{\max}^{D_{2n}}$ has no successor.

Regarding $w \in S_{2n}^{(j)}$, $2 \le j \le 4$, and $\mathsf{next}(w)$, we see that $w \prec \mathsf{next}(w)$. Hence we have only to prove that there is no Dyck word between w and $\mathsf{next}(w)$.

$w \in S_{2n}^{(2)}$: We assume that $(\exists v \in D_{2n})(w \prec v \prec \mathsf{next}(w)) \rightsquigarrow v = x\,\tilde{v}$, $x \in T^+$, with

$$
\begin{array}{lcl}
\underbrace{w_\alpha}_{\substack{\in T_\rfloor \\ \text{not maximal}}}\ \overbrace{\underbrace{w_{\alpha+1}\ldots w_\beta}_{\substack{\in T_\rfloor^\star \\ \text{maximal}}}\ w_{\max}^{D_{2i}}}^{\substack{\text{lexicographically} \\ \text{largest suffix for } x\,w_\alpha}} & \prec & \tilde{v} \\
& \prec & \underbrace{\mathsf{succ}(w,\alpha)}_{\in T_\rfloor}\ \overbrace{w_{\min}^{D_{2i}}\ \underbrace{\mathsf{min\_pred}(w,\alpha+1,\beta)}_{\substack{\in T_\rfloor^\star \\ \text{minimal}}}}^{\substack{\text{lexicographically} \\ \text{smallest suffix for } x\,\mathsf{succ}(w,\alpha)}}\ .
\end{array}
$$

Obviously, $w_\alpha \lessdot \mathsf{succ}(w,\alpha)$. Note that w_α and $\mathsf{succ}(w,\alpha)$ are the only possibilities for the first symbol in $\tilde{v}$. Further, we know that every $v \in D_{2n}$ has exactly n opening and n closing brackets.

Case 1: $\tilde{v} = w_\alpha\,\hat{v}$. $\rightsquigarrow w_{\alpha+1}\ldots w_\beta\, w_{\max}^{D_{2i}} \prec \hat{v}$. This is a contradiction, as w is the lexicographically largest Dyck word with the prefix $x\,w_\alpha$. Obviously, $\hat{v}$ must contain i opening brackets and $(\beta-\alpha+i)$ closing brackets, but all closing brackets in $w_{\alpha+1}\ldots w_\beta$ are maximal (so none of them can be substituted by a larger one), and there is no larger Dyck word in D_{2i} than $w_{\max}^{D_{2i}}$.

As each opening bracket is smaller than each closing bracket, a rearrangement of the brackets does not lead to a lexicographically larger Dyck word.

Thus such $\hat{v}$ does not exist.

Case 2: $\tilde{v} = \mathsf{succ}(w,\alpha)\,\hat{v}$. $\rightsquigarrow \hat{v} \prec w_{\min}^{D_{2i}}\,\mathsf{min\_pred}(w,\alpha+1,\beta)$. Similarly to Case 1, we can prove that $\mathsf{next}(w)$ is the smallest Dyck word with the prefix $x\,\mathsf{succ}(w,\alpha)$.

Hence such $\hat{v}$ does not exist.

$w \in S_{2n}^{(3)}$, $w \in S_{2n}^{(4)}$. These cases can be proved analogously to the previous case. □

Remark 2.2.

1. When regarding the definition of $\mathsf{next}(w)$ of $w \in D_{2n}$, we notice that we need some information on the relation $\mathcal{R}$:
   - Is a closing bracket maximal?
   - If a closing bracket is not maximal, what is its next closing bracket according to the relation $\mathcal{R}$ $\left(\widehat{=}\ \mathsf{succ}\right)$?
   - If a closing bracket is maximal, what is its minimal closing bracket according to the relation $\mathcal{R}$ $\left(\widehat{=}\ \mathsf{min\_pred}\right)$?

   In the next section, we will focus on the condition for the following:
   - This information can be obtained from the relation $\mathcal{R}$.
   - This information cannot be obtained from the relation $\mathcal{R}$, but it can be obtained by reading a part of the Dyck word—especially the opening bracket corresponding to that closing bracket.
2. If $n = 1$, the function next simplifies to

$$\mathsf{next}(w) = \begin{cases} \text{undefined} & \text{if} \quad w = w_{\max}^{D_2} \in S_2^{(1)}, \\ w_0\, \mathsf{succ}(w, 1) & \text{if} \quad w = w_0\, w_1 \in S_2^{(2)}, \\ & \quad (w_0, w_1) \in \mathcal{R},\ w_1 \text{ is not maximal}, \\ \mathsf{succ\_pair}(w_0\, w_1) & \text{if} \quad w = w_0\, w_1 \in S_2^{(3)}, \\ & \quad (w_0, w_1) \in \mathcal{R} \setminus \{\mathcal{R}_{\max}\},\ w_1 \text{ is maximal} \end{cases}$$
$$= \mathsf{succ\_pair}(w_0\, w_1).$$

Note that the function $\mathsf{min\_pred}$ is not needed for $n = 1$ because $S_2^{(4)} = \emptyset$.

Now, we are able to present the algorithmic definition of the function next. Regarding Theorem 2.3, we see that it has to read the Dyck word w from right to left. First, the algorithm reads all lexicographically maximal pairs of brackets $p_{\max}$ at the end of w. Then it has to read the string consisting of maximal closing brackets. Having read an opening bracket or a closing bracket that is not maximal, it finds the suffix to be changed and can generate the successor. We see that the function next uses the existence of the unique factorization according to the suffix of every Dyck word, cf. Lemma 2.2. The successor is computed for each of the sets defined in Lemma 2.1.

Sometimes there are one or more closing brackets in the suffix to be changed for which it is undecidable (in consideration of the suffix read only) whether the brackets are maximal or what the next or minimal closing brackets according to the order on the alphabet and the relation $\mathcal{R}$ are. If such a bracket is read, the function init is called. It reads to the left until the corresponding opening bracket is found (see Remark 1.1). The function init makes the information on this part of the Dyck word accessible to the functions succ and $\mathsf{min\_pred}$ in the algorithm.

Now, we are able to present the algorithm (see Algorithm 2) that generates all words in D_{2n} lexicographically. Note that $w = w_0 \dots w_{2n-1} \in D_{2n}$ in the algorithm.

3. On the length of the suffixes read and changed. In the preceding section, we noticed that, for the generation of the successor of a word $w \in D_{2n}$, according to the lexicographical order it might sometimes be necessary to read more than just the word's suffix to be changed (function init). In this section, we will show that this necessity depends on the relation $\mathcal{R}$.

Now, we formalize the condition for a simply generated Dyck language D_{2n}. We will see that this condition depends on $\mathcal{R}$ only and that we have to distinguish between the cases $n = 1$ and $n \geq 2$.

```
dyck word function next(w : dyck word)
  begin
    i := 2n − 1
    pairs := 0
    brackets := 0
    /⋆ ⋆ ⋆ ⋆ ⋆ find the suffix of w to be changed ⋆ ⋆ ⋆ ⋆ ⋆/
    /⋆ read w_max^{D_2pairs} ⋆/
    while i ≥ 1 and w_{i−1} w_i = p_max do
      begin
        pairs := pairs + 1
        i := i − 2
      end
    if i < 1 then
      begin
        /⋆ successor for w = w_max^{D_2n} ∈ S_2n^(1) does not exist ⋆/
        next := undefined
        return
      end
    /⋆ read all maximal closing brackets on the left of w_max^{D_2pairs} ⋆/
    while w_i ∈ T_] and no_succ(w_i) and min_pred_unique(w_i) do
      begin
        brackets := brackets + 1
        i := i − 1
      end
    if w_i ∈ T_] and (     not succ_unique(w_i)
                       or (no_succ(w_i) and not min_pred_unique(w_i))) then
      begin
        init(w, i)
        while w_i ∈ T_] and w_i is maximal do
          begin
            brackets := brackets + 1
            i := i − 1
          end
      end
    /⋆ ⋆ ⋆ ⋆ ⋆ change the suffix of w to generate the successor ⋆ ⋆ ⋆ ⋆ ⋆/
    if w_i ∈ T_] then
      begin
        /⋆ compute successor for w ∈ S_2n^(2) ⋆/
        w_i := succ(w, i)
        w_{2n−brackets} ... w_{2n−1}
          := min_pred(w, 2n − brackets − 2pairs, 2n − 1 − 2pairs)
        w_{i+1} ... w_{i+2pairs} := w_min^{D_2pairs}
       end
    else
      begin
        if w_i w_{i+1} ≠ p_max then
          begin
            /⋆ compute successor for w ∈ S_2n^(3) ⋆/
            w_{2n+1−brackets} ... w_{2n−1}
              := min_pred(w, 2n + 1 − brackets − 2pairs, 2n − 1 − 2pairs)
            w_i w_{i+1} := succ_pair(w_i w_{i+1})
            w_{i+1+2pairs} := w_{i+1}
            w_{i+1} ... w_{i+2pairs} := w_min^{D_2pairs}
          end
        else
          begin
            /⋆ compute successor for w ∈ S_2n^(4) ⋆/
            w_i := min_pred(w, i + 2)
            w_{2n+1−brackets} ... w_{2n−1}
              := min_pred(w, 2n + 1 − brackets − 2pairs, 2n − 1 − 2pairs)
            w_{i+1} ... w_{i+2+2pairs} := w_min^{D_2(pairs+1)}
          end
      end
    next := w
  end;
```

FIG. 4. *Algorithm* 2. *Successor function* next *for the lexicographical generation of Dyck words.*

$$\begin{pmatrix} & \vdots & & \vdots & \\ \cdots & m_{\kappa,\lambda} & 0 \cdots 0 & m_{\kappa,\lambda+\nu} & \cdots \\ & \vdots & & \vdots & \\ \cdots & m_{\kappa+\mu,\lambda} & \underbrace{0 \cdots 0}_{\geq 0} & m_{\kappa+\mu,\lambda+\nu} & \cdots \\ & \vdots & & \vdots & \end{pmatrix}$$

FIG. 5. *Matrix representation of a relation* $\mathcal{R}$ *(Case* a*).*

$$\begin{pmatrix} & \vdots & & \vdots & \\ 0 \cdots 0 & m_{\kappa,\lambda} & \cdots & m_{\kappa,\lambda+\nu} & 0 \cdots 0 \\ & \vdots & & \vdots & \\ \underbrace{0 \cdots 0}_{\geq 0} & m_{\kappa+\mu,\lambda} & \underbrace{\cdots}_{\geq 0} & m_{\kappa+\mu,\lambda+\nu} & \underbrace{0 \cdots 0}_{\geq 0} \\ & \vdots & & \vdots & \end{pmatrix}$$

FIG. 6. *Matrix representation of a relation* $\mathcal{R}$ *(Case* b*).*

THEOREM 3.1.
Case 1: $n = 1$.

D_2 *is simply generated*
$\iff \big(\forall\,]_j \in T_]\big)\big(\forall\, [_{i_1}, [_{i_2} \in \mathsf{open}(\,]_j)\big)\big(\mathsf{successor}(\,]_j, [_{i_1}) = \mathsf{successor}(\,]_j, [_{i_2})\big)$.

Case 2: $n \geq 2$.

D_{2n} *is simply generated* $\iff \big(\forall\,]_j \in T_]\big)\big(\forall\, [_{i_1}, [_{i_2} \in \mathsf{open}(\,]_j)\big)\big(\mathsf{close}([_{i_1}) = \mathsf{close}([_{i_2})\big)$.

Proof. We just outline the proof of the theorem; the formal proof is presented in [Li98] in detail.

First, let us take a look at some columns of two rows of the matrix representation M given in Figures 5 and 6. In both figures, we have $1 \leq \kappa < \kappa + \mu \leq |T_[|$ and $1 \leq \lambda < \lambda + \nu \leq |T_]|$. We distinguish between two cases, which we will refer to in the sketch of the proof.

Case a (see Figure 5):

$$\begin{pmatrix} m_{\kappa,\lambda} & m_{\kappa,\lambda+\nu} \\ m_{\kappa+\mu,\lambda} & m_{\kappa+\mu,\lambda+\nu} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \text{ or } \begin{pmatrix} m_{\kappa,\lambda} & m_{\kappa,\lambda+\nu} \\ m_{\kappa+\mu,\lambda} & m_{\kappa+\mu,\lambda+\nu} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Case b (see Figure 6):

$$\begin{pmatrix} m_{\kappa,\lambda} & m_{\kappa,\lambda+\nu} \\ m_{\kappa+\mu,\lambda} & m_{\kappa+\mu,\lambda+\nu} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \text{ or } \begin{pmatrix} m_{\kappa,\lambda} & m_{\kappa,\lambda+\nu} \\ m_{\kappa+\mu,\lambda} & m_{\kappa+\mu,\lambda+\nu} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

Let us remember Remark 2.2. Generating the successor $\mathsf{next}(w)$ of $w \in D_{2n}$, we need to decide whether a given closing bracket $w_i =]_j \in T_]$ is maximal. If it is not maximal, then we need to compute $\mathsf{succ}(w, i)$; otherwise, we need to compute $\mathsf{min\_pred}(w, i)$, but only if $n \geq 2$.

Case 1: $n = 1$. This case implies Case a (Figure 5).

$\Rightarrow$: We assume

$$\big(\exists\,]_j \in T_]\big)\big(\exists\, [_{i_1}, [_{i_2} \in \mathsf{open}(\,]_j)\big)\big(\mathsf{successor}(\,]_j, [_{i_1}) \neq \mathsf{successor}(\,]_j, [_{i_2})\big).$$

We have either Case i or Case ii.

Case i: $\mathsf{successor}(\,]_j, [_{i_1}) =]_a \neq]_b = \mathsf{successor}(\,]_j, [_{i_2})$.

Case ii: $\mathsf{successor}(\,]_j, [_{i_1}) =]_a \neq$ undefined $= \mathsf{successor}(\,]_j, [_{i_2})$.

In both cases, we can conclude that the Dyck language is not simply generated, since the condition for a language to be simply generated (see Definition 1.6) is not fulfilled.

$\Leftarrow$: We assume that D_2 is not simply generated, i.e.,

$$\big(\exists w, w' \in D_2\big)\big(\mathsf{old}(w) = \vartheta\, \mathsf{old}(w'), \vartheta \in T^\star \rightsquigarrow (\vartheta, \mathsf{new}(w)) \neq (\varepsilon, \mathsf{new}(w'))\big).$$

Again, we have two cases.

Case i: $\vartheta \neq \varepsilon$.

Case ii: $\vartheta = \varepsilon \ \wedge\ \mathsf{new}(w) \neq \mathsf{new}(w')$.

In either case, we obtain a contradiction to

$$\big(\forall\,]_j \in T_]\big)\big(\forall\, [_{i_1}, [_{i_2} \in \mathsf{open}(\,]_j)\big)\big(\mathsf{successor}(\,]_j, [_{i_1}) = \mathsf{successor}(\,]_j, [_{i_2})\big).$$

Case 2: $n \geq 2$. This case implies Case a (Figure 5) or Case b (Figure 6); it can be proved in a way analogous to the proof of Case 1. □

Remark 3.1.

1. From Theorem 3.1 it follows immediately that a Dyck language D_{2n} is simply generated for $n \geq 1$ if $|\mathsf{open}(\,]_j)| = 1$, $1 \leq j \leq |T_]|$.
2. In order to compute $\mathsf{next}(w)$ of $w \in D_{2n}$, we have to read a suffix of w. If D_{2n} is simply generated, we have to change exactly the brackets we need to read in order to generate the successor of $w \in D_{2n}$. In that case, the information required by the functions succ, min_pred, no_succ, succ_unique, and min_pred_unique can be directly deduced from the relation $\mathcal{R}$. This is in contrast to a language not simply generated, whereas—for at least one word —a part of the common prefix of the Dyck word and its successor has to be inspected.
3. The familiar Dyck languages with the following relations are simply generated:
   - $\mathcal{R} = \{([_1,]_1)\}$, i.e., with one pair of brackets,
   - $\mathcal{R} = \{([_1,]_1), \ldots, ([_t,]_t)\}$, i.e., with t pairs of brackets, $t \geq 1$, where every opening (resp., closing) bracket corresponds to one closing (resp., opening) bracket only (cf. [Ha78, p. 313], [Li96]), and
   - $\mathcal{R} = \{([_1,]_1), \ldots, ([_1,]_r), ([_2,]_1), \ldots, ([_2,]_r), \ldots, ([_l,]_1), \ldots, ([_l,]_r)\}$, i.e., with $l \cdot r$ pairs of brackets, $l, r \geq 1$, where each opening (resp., closing) bracket corresponds to each closing (resp., opening) bracket.
4. Let

$$\widetilde{M} := \begin{pmatrix} A^{(1)} & & & & \\ & A^{(2)} & & 0 & \\ & & \ddots & & \\ & 0 & & A^{(k-1)} & \\ & & & & A^{(k)} \end{pmatrix}$$

be the *normal form* of a matrix M. Here, $A^{(l)}$, $1 \leq l \leq k$, are submatrices.

```
boolean function simply_generated ()
    begin
        simply_generated := true
        /⋆ check for all closing brackets ]_j, ... ⋆/
        for j := 1 to |T_]| do
            begin
                /⋆ ... if successor(]_j, [_i) is the same for all corresponding [_i ⋆/
                k := 0
                for i := 1 to |T_[| do
                    if m_{i,j} = 1 and k = 0 then
                        k := search_next(i, j)
                    else if m_{i,j} = 1 and k ≠ search_next(i, j) then
                        begin
                            simply_generated := false
                            return
                        end
            end
    end;
```

FIG. 7. *Algorithm* 3. *Function that checks whether the language D_2 is simply generated.*

- If $n = 1$, the first row of $A^{(l)}$ consists of 1's only. The following rows have the form

$$\underbrace{0 \cdots 0}_{\geq 0} \underbrace{1 \cdots 1}_{\geq 1},$$

  where the number of 1's is monotonically decreasing from the first to the last row. For example, the matrices

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \text{ and } \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

  have that property.
- In the case in which $n \geq 2$, each entry in $A^{(l)}$ is equal to 1.

The matrix representation M of the Dyck language D_{2n} with relation $\mathcal{R}$ can be transformed into its normal form $\widetilde{M}$ by permutating rows and columns iff D_{2n} is simply generated.

Example 3.1. D_{2n} with relation $\mathcal{R} = \{([_1,]_1), ([_2,]_2)\}$ of Example 1.1 is simply generated for $n \geq 1$, as $|\mathsf{open}(\,]_1)| = 1$ and $|\mathsf{open}(\,]_2)| = 1$.

For the language D_{2n} with relation $\mathcal{R} = \{([_1,]_1), ([_1,]_2), ([_2,]_2)\}$ of Example 1.2, we find $\mathsf{open}(\,]_1) = \{[_1\}$ and $\mathsf{open}(\,]_2) = \{[_1, [_2\}$. As $\mathsf{successor}(\,]_2, [_1) =$ undefined $= \mathsf{successor}(\,]_2, [_2)$, D_{2n} is simply generated for $n = 1$. Since $\mathsf{close}([_1) = \{\,]_1,]_2\} \neq \{\,]_2\} = \mathsf{close}([_2)$, D_{2n} is not simply generated for $n \geq 2$.

Now, one can determine how to decide algorithmically whether D_{2n} is simply generated or not. As we have seen, the property of being simply generated depends on the relation $\mathcal{R}$ only; we have to distinguish between $n = 1$ and $n \geq 2$. Let us have a look at an algorithm that decides whether D_{2n} is simply generated or not. We discuss the algorithm for $n = 1$; an algorithm for $n \geq 2$ is similar.

Algorithm 3 checks if the language D_2 is simply generated. It operates on the matrix representation of $\mathcal{R}$.

The function simply_generated uses another function that computes $\mathsf{successor}(\,]_j, [_i)$ for any $([_i,]_j) \in \mathcal{R}$ by looking up the column of the next 1 on the right side of the

```
integer function search_next (i, j : integer)
    begin
        search_next := j + 1
        while search_next ≤ |T⌋| and m_{i,search_next} = 0 do
            search_next := search_next + 1
    end;
```

FIG. 8. *Algorithm* 4. *Function that looks up the column of the next* 1 *on the right of an entry in* M.

entry for $([_i,]_j)$ in M in the same row. If such a 1 does not exist, $\left|T_{\rfloor}\right| + 1$ will be returned, which means $\mathsf{successor}(\,]_j, [_i) =$ undefined. The function search_next is defined in Algorithm 4.

Note that each entry in the matrix M is regarded twice at most. So, it can be checked in $\mathcal{O}\big(\left|T_{\lfloor}\right| \left|T_{\rfloor}\right|\big)$ if the language D_2 is simply generated. The same fact holds for the case in which $n \geq 2$. Hence the amount of time to decide whether the language is simply generated is constant with respect to n, i.e., independent of the number of pairs of brackets in the words of the language.

4. Analysis of the algorithm. In this section, we analyze the length of the suffix to be changed in order to compute the successor $\mathsf{next}(w)$ of a Dyck word $w \in D_{2n}$. Remember that for every word the suffix to be changed is equal to the suffix to be read if D_{2n} is simply generated. If D_{2n} is not simply generated, then there is at least one Dyck word $w \in D_{2n} \setminus \{w_{\max}^{D_{2n}}\}$ for which an algorithm has to inspect the common prefix $\mathsf{pre}(w)$ of w and $\mathsf{next}(w)$. So, the length of the suffix to be read is larger than the length of the suffix to be changed.

Let $X_{\mathsf{sufl}}(D_{2n})$ be the random variable that describes the length of the suffix to be changed. We proved in section 2 that the function generate generates the Dyck words in D_{2n} with respect to the lexicographical order $\prec$. Thus every word is generated exactly once. Further, the function next reads the words from right to left. Under these conditions, the sth moments, $s \geq 1$, about the origin of the random variable $X_{\mathsf{sufl}}(D_{2n})$ are given by [Ke98]:

$$\mathbb{E}[X_{\mathsf{sufl}}^s(D_{2n})] := 1 + |D_{2n}|^{-1} \sum_{k=1}^{2n-1} [(k+1)^s - k^s] \; |\mathsf{INIT}_{2n-k}(D_{2n})| \,. \tag{4.1}$$

Here, $\mathsf{INIT}_k(D_{2n}) := \mathsf{INIT}(D_{2n}) \cap T^k$ denotes the *set of all prefixes of length* k *appearing in words belonging to* D_{2n}; thereby, the *set of all prefixes appearing in words belonging to* D_{2n} is defined by $\mathsf{INIT}(D_{2n}) := \{u \in T^\star \mid (\exists v \in T^\star)\,(uv \in D_{2n})\}$.

Now, our interest is $|\mathsf{INIT}_k(D_{2n})|$, $1 \leq k < 2n$. For that purpose we consider the well-known one-to-one correspondence between the Dyck language and paths on the lattice given in Figure 9.

Thereby, an *up-segment* $\nearrow$ (resp., *down-segment* $\searrow$) corresponds to an opening (resp., closing) bracket. Since we have a correspondence between opening and closing brackets (see Remark 1.1), there must also be a correspondence between up-segments and down-segments of a path in the lattice. To each up-segment of any path there is a corresponding down-segment and vice versa: The down-segment (resp., up-segment) corresponding to an up-segment (resp., down-segment) is the next segment on the right (resp., left) side in the same row.

Thus each Dyck word of length $2n$ corresponds to a path from $(0, 0)$ to $(2n, 0)$. The segments are labeled by the symbols of the alphabet; the labels of up-segments (resp.,

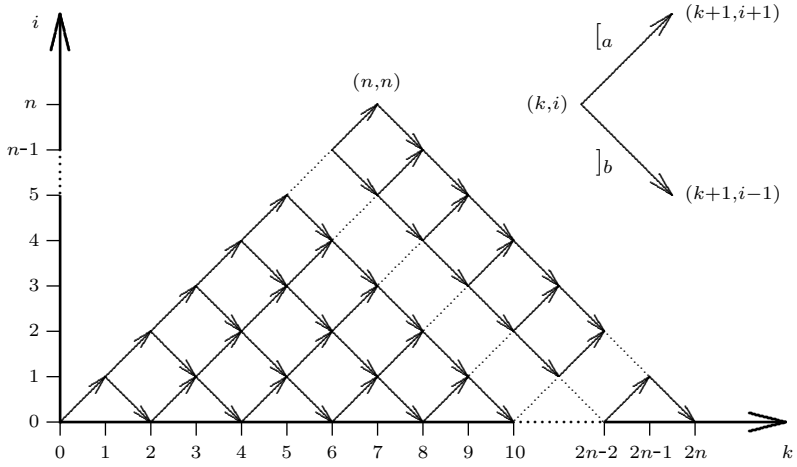


FIG. 9. *One-to-one correspondence between Dyck words of length* $2n$ *and the paths from* $(0,0)$ *to* $(2n,0)$

down-segments) are opening (resp., closing) brackets. Obviously, an up-segment and its corresponding down-segment must be labeled by some $[_a$ and $]_b$ with $\big([_a,]_b\big) \in \mathcal{R}$.

With the number of paths from $(0,0)$ to (k,i) in the lattice in Figure 9 given by the ballot number [CRS71]

$$\mathsf{p}(k,i) = \binom{k}{\frac{1}{2}(k-i)} - \binom{k}{\frac{1}{2}(k-i)-1}, \tag{4.2}$$

we have

$$|\mathsf{INIT}_k(D_{2n})| = \sum_{\substack{0\le i\le \min\{k,2n-k\}\\ k+i\ \equiv\ 0 \bmod 2}} |\mathcal{R}|^{\frac{1}{2}(k-i)}\ \big|T_{[}\big|^{i}\ \mathsf{p}(k,i)\ . \tag{4.3}$$

This formula can be found in [Ke96].

Distinguishing between the terms for even k and those for odd k in (4.3), we immediately obtain by inserting (4.2)

$$\begin{aligned} |\mathsf{INIT}_k(D_{2n})| \quad = \quad & \sum_{i=0}^{\min\{\lfloor \frac{k}{2}\rfloor,\, n-\lfloor \frac{k+1}{2}\rfloor\}} |\mathcal{R}|^{\lfloor \frac{k}{2}\rfloor - i}\ \big|T_{[}\big|^{2i+k-2\lfloor \frac{k}{2}\rfloor} \\ & \times \left[\binom{k}{\lfloor \frac{k}{2}\rfloor - i} - \binom{k}{\lfloor \frac{k}{2}\rfloor - i - 1}\right]\ . \end{aligned} \tag{4.4}$$

By (4.1), $|D_{2n}| = |\mathsf{INIT}_{2n}(D_{2n})|$, and (4.4), we get

$$\begin{aligned} \mathbb{E}[X^s_{\mathsf{sufl}}(D_{2n})] = |D_{2n}|^{-1} \sum_{k=0}^{2n-1} [(k+1)^s - k^s] & \sum_{i=0}^{\min\{\lfloor \frac{2n-k}{2}\rfloor,\, n-\lfloor \frac{2n-k+1}{2}\rfloor\}} |\mathcal{R}|^{\lfloor \frac{2n-k}{2}\rfloor - i} \\ \times \big|T_{[}\big|^{2i+2n-k-2\lfloor \frac{2n-k}{2}\rfloor} & \left[\binom{2n-k}{\lfloor \frac{2n-k}{2}\rfloor - i} - \binom{2n-k}{\lfloor \frac{2n-k}{2}\rfloor - i - 1}\right]\ . \end{aligned}$$

Splitting the first sum into two parts and letting the index of the second sum go to infinity, we obtain with $|D_{2n}| = \frac{1}{n+1}\binom{2n}{n}|\mathcal{R}|^n$ after rearranging the sums and applying

some simplifications

$$\mathbb{E}[X^s_{\mathsf{sufl}}(D_{2n})] = \frac{n+1}{\binom{2n}{n}}\left[F^{(1)}_{s,|\mathcal{R}|,|T_{\lfloor}|}(n) + F^{(2)}_{s,|\mathcal{R}|,|T_{\lfloor}|}(n) - F^{(3)}_{s,|\mathcal{R}|,|T_{\lfloor}|}(n) - F^{(4)}_{s,|\mathcal{R}|}(n)\right],$$

where

$$F^{(1)}_{s,a,b}(n) := \sum_{i\geq 0} a^{-i}\, b^{2i} \sum_{k=0}^{n} \left[(2k+1)^s - (2k)^s\right] a^{-k} \times \left[\binom{2n-2k}{n-k-i} - \binom{2n-2k}{n-k-i-1}\right],$$

$$F^{(2)}_{s,a,b}(n) := \sum_{i\geq 0} a^{-i-1}\, b^{2i+1} \sum_{k=0}^{n} \left[(2k+2)^s - (2k+1)^s\right] a^{-k} \times \left[\binom{2n-2k-1}{n-k-i-1} - \binom{2n-2k-1}{n-k-i-2}\right],$$

$$F^{(3)}_{s,a,b}(n) := \sum_{i\geq 0} a^{-i-1}\, b^{2i+2} \sum_{k=0}^{n} \left[(k+1)^s - k^s\right] a^{-k}\, b^k \times \left[\binom{2n-k}{n-k-i-1} - \binom{2n-k}{n-k-i-2}\right],$$

$$F^{(4)}_{s,a}(n) := \frac{(2n+1)^s - (2n)^s}{a^n}\ .$$

In order to gain an asymptotic for $n \to \infty$ for $\mathbb{E}[X^s_{\mathsf{sufl}}(D_{2n})]$, we need to study the asymptotic behavior of $F^{(1)}_{s,a,b}(n)$, $F^{(2)}_{s,a,b}(n)$, $F^{(3)}_{s,a,b}(n)$, and $F^{(4)}_{s,a}(n)$; that is why we consider the generating functions for these functions in Lemma 4.1.

LEMMA 4.1. *Let s, a, $b \in \mathbb{N}$, and $u := \sqrt{1-4z}$. The generating functions of $F^{(1)}_{s,a,b}(n)$, $F^{(2)}_{s,a,b}(n)$, $F^{(3)}_{s,a,b}(n)$, and $F^{(4)}_{s,a}(n)$ are given by*

$$G^{(1)}_{s,a,b}(z) := \sum_{n\geq 0} F^{(1)}_{s,a,b}(n)\, z^n = \frac{2}{(1+u)\left(1-\frac{b^2(1-u)^2}{4az}\right)} \times \left(\frac{1}{1-\frac{z}{a}} + \sum_{j=1}^{s-1}\binom{s}{j} 2^j\, \mathsf{A}_j\left(\frac{z}{a}\right) \frac{1}{\left(1-\frac{z}{a}\right)^{j+1}}\right),$$

$$G^{(2)}_{s,a,b}(z) := \sum_{n\geq 0} F^{(2)}_{s,a,b}(n)\, z^n = \frac{b(1-u)}{a(1+u)\left(1-\frac{b^2(1-u)^2}{4az}\right)} \left(\frac{(-1)^{s+1}}{1-\frac{z}{a}} + \frac{a}{z}\sum_{j=1}^{s-1}\binom{s}{j} 2^j\, (-1)^{s-j+1}\, \mathsf{A}_j\left(\frac{z}{a}\right) \frac{1}{\left(1-\frac{z}{a}\right)^{j+1}}\right),$$

$$G^{(3)}_{s,a,b}(z) := \sum_{n\geq 0} F^{(3)}_{s,a,b}(n)\, z^n = \frac{4b}{(1+u)^2\left(1-\frac{b^2(1-u)}{a(1+u)}\right)} \mathsf{A}_s\left(\frac{b(1-u)}{2a}\right) \frac{1}{\left(1-\frac{b(1-u)}{2a}\right)^s},$$

$$G^{(4)}_{s,a}(z) := \sum_{n\geq 0} F^{(4)}_{s,a}(n)\, z^n = \frac{1}{1-\frac{z}{a}} + \sum_{j=1}^{s-1}\binom{s}{j} 2^j\, \mathsf{A}_j\left(\frac{z}{a}\right) \frac{1}{\left(1-\frac{z}{a}\right)^{j+1}}\ .$$

Here, $\mathsf{A}_l(x)$ *denotes the* l*th Eulerian polynomial* [Ke84, p. 214] *with*

$$\mathsf{A}_l(x) = (1-x)^{l+1} \sum_{m\geq 0} m^l x^m = x \sum_{i=1}^{l} i! \left\{ {l \atop i} \right\} (x-1)^{l-i},$$

where $\left\{ {n \atop k} \right\}$ *stands for the Stirling number of the second kind* [GKP94, p. 258].

Proof. In order to find the generating function for $F^{(1)}_{s,a,b}(n)$, we consider the convolution of two functions first.

$$\left(\sum_{\mu\geq 0} [(2\mu+1)^s - (2\mu)^s]\, a^{-\mu}\, z^\mu \right) \left(\sum_{\lambda\geq 0} \left[\binom{2\lambda}{\lambda-i} - \binom{2\lambda}{\lambda-i-1} \right] z^\lambda \right)$$

$$= \sum_{\kappa\geq 0} z^\kappa \sum_{\nu=0}^{\kappa} [(2\nu+1)^s - (2\nu)^s]\, a^{-\nu} \left[\binom{2\kappa-2\nu}{\kappa-\nu-i} - \binom{2\kappa-2\nu}{\kappa-\nu-i-1} \right]. \tag{4.5}$$

With $u = \sqrt{1-4z} \rightsquigarrow 4z = (1+u)(1-u)$ and the identity

$$\sum_{m\geq 0} \binom{2m+\alpha}{m} z^m = \frac{1}{\sqrt{1-4z}} \left(\frac{1-\sqrt{1-4z}}{2z} \right)^\alpha = \frac{1}{u} \left(\frac{1-u}{2z} \right)^\alpha,\ \alpha \in \mathbb{N}_0 \tag{4.6}$$

[GKP94, p. 203], a straightforward computation leads to

$$\sum_{\lambda\geq 0} \left[\binom{2\lambda}{\lambda-i} - \binom{2\lambda}{\lambda-i-1} \right] z^\lambda = \frac{2}{1+u} \left(\frac{(1-u)^2}{4z} \right)^i. \tag{4.7}$$

In consideration of (4.5) and (4.7), we obtain

$$\begin{aligned} G^{(1)}_{s,a,b}(z) &= \sum_{i\geq 0} a^{-i}\, b^{2i} \left(\sum_{\lambda\geq 0} \left[\binom{2\lambda}{\lambda-i} - \binom{2\lambda}{\lambda-i-1} \right] z^\lambda \right) \\ &\qquad \times \left(\sum_{\mu\geq 0} [(2\mu+1)^s - (2\mu)^s]\, a^{-\mu}\, z^\mu \right) \\ &= \frac{2}{(1+u)\left(1 - \frac{b^2(1-u)^2}{4az}\right)} \left(\sum_{\mu\geq 0} [(2\mu+1)^s - (2\mu)^s]\, a^{-\mu}\, z^\mu \right) \end{aligned} \tag{4.8}$$

by the expansion of the geometric series. Now, let us have a closer look at the sum in (4.8). Splitting off the first term, applying the binomial theorem to $(2\mu+1)^s$, and using again the expansion of the geometric series yield

$$\begin{aligned} \sum_{\mu\geq 0} [(2\mu+1)^s - (2\mu)^s]\, a^{-\mu}\, z^\mu &= \sum_{\mu\geq 0} \left(\frac{z}{a}\right)^\mu + \sum_{j=1}^{s-1} \binom{s}{j} 2^j \sum_{\mu\geq 0} \mu^j \left(\frac{z}{a}\right)^\mu \\ &= \frac{1}{1-\frac{z}{a}} + \sum_{j=1}^{s-1} \binom{s}{j} 2^j\, \mathsf{A}_j\left(\frac{z}{a}\right) \frac{1}{\left(1-\frac{z}{a}\right)^{j+1}}. \end{aligned} \tag{4.9}$$

Inserting the expression (4.9) into (4.8) results in the generating function $G^{(1)}_{s,a,b}(z)$ for the numbers $F^{(1)}_{s,a,b}(n)$ stated in the lemma. $G^{(2)}_{s,a,b}(z)$ can be computed analogously.

For the generating function $G^{(3)}_{s,a,b}(z)$, we consider the function

$$H_{s,c,d}(z) := \sum_{n\geq 0} z^n \sum_{k=0}^{n} [(k+1)^s - k^s] \binom{2n-k}{n+c} d^k, \quad c \in \mathbb{N}_0,\ d > 0.$$

A rearrangement of the terms of $H_{s,c,d}(z)$ and the application of the identity $\binom{n}{k} = \binom{n}{n-k}$, n, $k \in \mathbb{N}_0$, results with (4.6) in

$$\begin{aligned} H_{s,c,d}(z) &= \sum_{k\geq 0} [(k+1)^s - k^s]\, d^k\, z^{k+c} \sum_{n\geq 0} z^n \binom{2n+k+2c}{n} \\ &= \frac{1}{u}\left(\frac{1-u}{1+u}\right)^c \sum_{k\geq 0} [(k+1)^s - k^s] \left(\frac{d(1-u)}{2}\right)^k. \end{aligned}$$

Moreover, a simple computation shows that

$$\sum_{k\geq 0} [(k+1)^s - k^s]\, y^k = \mathsf{A}_s(y)\, \frac{1}{y(1-y)^s}$$

holds; applying this identity, we obtain

$$H_{s,c,d}(z) = \frac{1}{u}\left(\frac{1-u}{1+u}\right)^c \mathsf{A}_s\left(\frac{d(1-u)}{2}\right) \frac{1}{\frac{d(1-u)}{2}\left(1-\frac{d(1-u)}{2}\right)^s}. \tag{4.10}$$

Now, by (4.10), we get after simplifications

$$\begin{aligned} G^{(3)}_{s,a,b}(z) &= \sum_{n\geq 0} z^n \sum_{i\geq 0} \left(\frac{b^2}{a}\right)^{i+1} \sum_{k=0}^{n} [(k+1)^s - k^s] \left(\frac{b}{a}\right)^k \\ &\qquad \times \left[\binom{2n-k}{n+i+1} - \binom{2n-k}{n+i+2}\right] \\ &= \frac{4b}{(1+u)^2\left(1-\frac{b^2(1-u)}{a(1+u)}\right)} \mathsf{A}_s\left(\frac{b(1-u)}{2a}\right) \frac{1}{\left(1-\frac{b(1-u)}{2a}\right)^s}. \end{aligned}$$

$G^{(4)}_{s,a}(z)$ can be calculated similarly to $G^{(1)}_{s,a,b}(z)$. □

With this lemma, we are able to formalize the following theorem.

THEOREM 4.2. *Let all $w \in D_{2n}$ be equally likely. The sth moments, $s \geq 1$, about the origin of the random variable $X_{\mathsf{sufl}}(D_{2n})$ are constant. They are given by*

$$\mathbb{E}[X^s_{\mathsf{sufl}}(D_{2n})] \sim C_{s,|\mathcal{R}|,|T_{\lceil}|},\ n \to \infty,\ \textit{where}$$

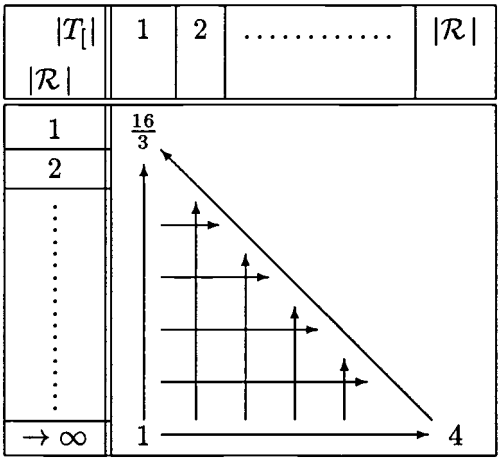
$$C_{s,|\mathcal{R}|,|T_\llcorner|} := \frac{|\mathcal{R}|}{\left(|\mathcal{R}| - |T_\llcorner|^2\right)^2} \Bigg[\sum_{j=1}^{s-1} \binom{s}{j} 2^j \, \mathsf{A}_j\left(\frac{1}{4|\mathcal{R}|}\right) \frac{|\mathcal{R}| + |T_\llcorner|^2 + 4|\mathcal{R}||T_\llcorner|(-1)^{s-j+1}}{\left(1 - \frac{1}{4|\mathcal{R}|}\right)^{j+1}}$$
$$- \frac{2^s |\mathcal{R}|^{s-1}|T_\llcorner|}{(2|\mathcal{R}| - |T_\llcorner|)^{s+1}} \Bigg[2|\mathcal{R}|\left(4|\mathcal{R}|^2 + (s-2)|\mathcal{R}||T_\llcorner| - s|T_\llcorner|^3\right) \mathsf{A}_s\left(\frac{|T_\llcorner|}{2|\mathcal{R}|}\right)$$
$$+ |T_\llcorner|\left(2|\mathcal{R}| - |T_\llcorner|\right)\left(|\mathcal{R}| - |T_\llcorner|^2\right) \mathsf{A}_s'\left(\frac{|T_\llcorner|}{2|\mathcal{R}|}\right) \Bigg]$$
$$+ \frac{4|\mathcal{R}|\left(|\mathcal{R}| + |T_\llcorner|^2 - |T_\llcorner|(-1)^s\right)}{4|\mathcal{R}| - 1} \Bigg].$$

Proof. Using the results of the preceding lemma, we get the following formula after a simple computation:

$$\begin{aligned}
&\mathbb{E}[X_{\mathsf{sufl}}^s(D_{2n})] \\
&= \frac{n+1}{\binom{2n}{n}} [z^n] \left\{ G_{s,|\mathcal{R}|,|T_\llcorner|}^{(1)}(z) + G_{s,|\mathcal{R}|,|T_\llcorner|}^{(2)}(z) - G_{s,|\mathcal{R}|,|T_\llcorner|}^{(3)}(z) - G_{s,|\mathcal{R}|}^{(4)}(z) \right\} \\
&= \frac{n+1}{\binom{2n}{n}} [z^n] \Bigg\{ \frac{1}{(1+u)\left(1 - \frac{|T_\llcorner|^2(1-u)}{|\mathcal{R}|(1+u)}\right)} \Bigg[\left(2 - (1+u)\left(1 - \frac{|T_\llcorner|^2(1-u)}{|\mathcal{R}|(1+u)}\right)\right) \\
&\quad \times \left(\frac{1}{1 - \frac{z}{|\mathcal{R}|}} + \sum_{j=1}^{s-1} \binom{s}{j} 2^j \, \mathsf{A}_j\left(\frac{z}{|\mathcal{R}|}\right) \frac{1}{\left(1 - \frac{z}{|\mathcal{R}|}\right)^{j+1}} \right) + (1-u)\frac{|T_\llcorner|}{|\mathcal{R}|} \\
&\quad \times \left(\frac{(-1)^{s+1}}{1 - \frac{z}{|\mathcal{R}|}} + \frac{|\mathcal{R}|}{z} \sum_{j=1}^{s-1} \binom{s}{j} 2^j \, (-1)^{s-j+1} \, \mathsf{A}_j\left(\frac{z}{|\mathcal{R}|}\right) \frac{1}{\left(1 - \frac{z}{|\mathcal{R}|}\right)^{j+1}} \right) \\
&\quad - \frac{4|T_\llcorner|}{(1+u)} \mathsf{A}_s\left(\frac{|T_\llcorner|(1-u)}{2|\mathcal{R}|}\right) \frac{1}{\left(1 - \frac{|T_\llcorner|(1-u)}{2|\mathcal{R}|}\right)^s} \Bigg] \Bigg\}.
\end{aligned}$$

Now, we are looking for the singularity of smallest modulus that is not equal to 0. We find candidates for singularities at $z = \frac{1}{4}$, $z = |\mathcal{R}|$, $z = \frac{|\mathcal{R}||T_\llcorner|^2}{(|\mathcal{R}|+|T_\llcorner|^2)^2}$, and $z = \frac{|\mathcal{R}|(|T_\llcorner|-|\mathcal{R}|)}{|T_\llcorner|^2}$. We do not have to take $z = |\mathcal{R}| > \frac{1}{4}$ into account. As the expansions of $G_{s,|\mathcal{R}|,|T_\llcorner|}^{(1)}(z) + G_{s,|\mathcal{R}|,|T_\llcorner|}^{(2)}(z) - G_{s,|\mathcal{R}|,|T_\llcorner|}^{(3)}(z) - G_{s,|\mathcal{R}|}^{(4)}(z)$ around $z = \frac{|\mathcal{R}||T_\llcorner|^2}{(|\mathcal{R}|+|T_\llcorner|^2)^2}$ and $z = \frac{|\mathcal{R}|(|T_\llcorner|-|\mathcal{R}|)}{|T_\llcorner|^2}$ result in Taylor series, i.e., neither $z = \frac{|\mathcal{R}||T_\llcorner|^2}{(|\mathcal{R}|+|T_\llcorner|^2)^2}$ nor $z = \frac{|\mathcal{R}|(|T_\llcorner|-|\mathcal{R}|)}{|T_\llcorner|^2}$ is a singularity, the singularity nearest to the origin is given by $z = \frac{1}{4}$. Expanding $G_{s,|\mathcal{R}|,|T_\llcorner|}^{(1)}(z) + G_{s,|\mathcal{R}|,|T_\llcorner|}^{(2)}(z) - G_{s,|\mathcal{R}|,|T_\llcorner|}^{(3)}(z) - G_{s,|\mathcal{R}|}^{(4)}(z)$ around $y := 1 - 4z$ yields

TABLE 1
Values for $C_{1,|\mathcal{R}|,|T_{[}|}$, $1 \le |T_{[}| \le |\mathcal{R}|, |\mathcal{R}| \to \infty$.



$$G^{(1)}_{s,|\mathcal{R}|,|T_{[}|}(z) + G^{(2)}_{s,|\mathcal{R}|,|T_{[}|}(z) - G^{(3)}_{s,|\mathcal{R}|,|T_{[}|}(z) - G^{(4)}_{s,|\mathcal{R}|}(z)$$
$$= C'_{s,|\mathcal{R}|,|T_{[}|} - 2\,C_{s,|\mathcal{R}|,|T_{[}|}\,\sqrt{y} + \mathcal{O}(y),$$

where $C_{s,|\mathcal{R}|,|T_{[}|}$ is given in the theorem and $C'_{s,|\mathcal{R}|,|T_{[}|} = \mathcal{O}(1)$ is another constant. An application of Darboux's method [GK82] immediately gives the asymptotic behavior of the sth moments of the random variable $X_{\mathsf{sufl}}(D_{2n})$ stated in the theorem with $\frac{1}{n+1}\binom{2n}{n} \sim \frac{4^n}{\sqrt{\pi} n^{\frac{3}{2}}}$, $n \to \infty$, obtained by Stirling's formula [GKP94, p. 454]. □

COROLLARY 4.3. *For the random variable* $X_{\mathsf{sufl}}(D_{2n})$*, we obtain the mean value* $\mu_{\mathsf{sufl}}(D_{2n}) = \mathbb{E}[X^1_{\mathsf{sufl}}(D_{2n})]$ *and the variance* $\sigma^2_{\mathsf{sufl}}(D_{2n}) = \mathbb{E}[X^2_{\mathsf{sufl}}(D_{2n})] - \mathbb{E}[X^1_{\mathsf{sufl}}(D_{2n})]^2$ *for* $n \to \infty$:

$$\mu_{\mathsf{sufl}}(D_{2n}) \sim \frac{16|\mathcal{R}|^3}{\big(4|\mathcal{R}|-1\big)\big(2|\mathcal{R}|-|T_{[}|\big)^2},$$
$$\sigma^2_{\mathsf{sufl}}(D_{2n}) \sim \frac{16|\mathcal{R}|^3\big(16|\mathcal{R}|^2|T_{[}| + 12|\mathcal{R}|^2 - 12|\mathcal{R}||T_{[}|^2 - 20|\mathcal{R}||T_{[}| + 7|T_{[}|^2\big)}{\big(4|\mathcal{R}|-1\big)^2\big(2|\mathcal{R}|-|T_{[}|\big)^4}\ .$$

Remark 4.1. Now, we focus on $\mu_{\mathsf{sufl}}(D_{2n})$. According to Theorem 4.2, we have $\mu_{\mathsf{sufl}}(D_{2n}) \sim C_{1,|\mathcal{R}|,|T_{[}|}$, $n \to \infty$. A moment's reflection shows that

$$C_{1,1,1} = \frac{16}{3},$$
$$C_{1,|\mathcal{R}|,1} = \frac{16|\mathcal{R}|^3}{\big(4|\mathcal{R}|-1\big)\big(2|\mathcal{R}|-1\big)^2} = 1 + \mathcal{O}\left(\frac{1}{|\mathcal{R}|}\right), \quad |\mathcal{R}| \to \infty,$$
$$C_{1,|\mathcal{R}|,|\mathcal{R}|} = \frac{16|\mathcal{R}|}{\big(4|\mathcal{R}|-1\big)} = 4 + \mathcal{O}\left(\frac{1}{|\mathcal{R}|}\right), \quad |\mathcal{R}| \to \infty,$$

hold. We further find with $1 \le |T_{[}| \le |\mathcal{R}|$

$$C_{1,|\mathcal{R}|,|T_{[}|} > C_{1,|\mathcal{R}|+1,|T_{[}|}, \tag{4.11}$$

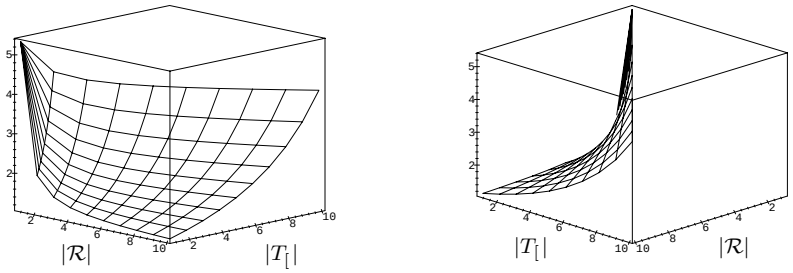


FIG. 10. *Two views of* $C_{1,|\mathcal{R}|,|T_{[}|}$ *for* $1 \le |T_{[}| \le |\mathcal{R}| \le 10$

TABLE 2
Exact and asymptotical values for $\mu_{\mathsf{sufl}}(D_{2n})$ *and* $\sigma_{\mathsf{sufl}}(D_{2n})$ *for relations with* $1 \le |T_{[}| \le |\mathcal{R}| \le 5$.

$\lvert\mathcal{R}\rvert \downarrow$	$n \downarrow$	$\lvert T_{[}\rvert \rightarrow$	1	2	3	4	5
1	10	$\mu_{\mathsf{sufl}}(D_{2n})$	4.91176				
	100		5.28127				
	$\to\infty$		5.33333				
	10	$\sigma^2_{\mathsf{sufl}}(D_{2n})$	3.73227				
	100		5.08069				
	$\to\infty$		5.33333				
2	10	$\mu_{\mathsf{sufl}}(D_{2n})$	2.04358	4.08490			
	100		2.03273	4.51384			
	$\to\infty$		2.03175	4.57143			
	10	$\sigma^2_{\mathsf{sufl}}(D_{2n})$	1.79522	2.65130			
	100		1.77458	4.29137			
	$\to\infty$		1.77375	4.57143			
3	10	$\mu_{\mathsf{sufl}}(D_{2n})$	1.58401	2.39785	3.86900		
	100		1.57212	2.44870	4.30531		
	$\to\infty$		1.57091	2.45455	4.36364		
	10	$\sigma^2_{\mathsf{sufl}}(D_{2n})$	0.97310	1.94598	2.40665		
	100		0.93482	2.20012	4.07994		
	$\to\infty$		0.93112	2.23140	4.36364		
4	10	$\mu_{\mathsf{sufl}}(D_{2n})$	1.40361	1.88379	2.61626	3.76953	
	100		1.39417	1.89503	2.71870	4.20811	
	$\to\infty$		1.39320	1.89630	2.73067	4.26667	
	10	$\sigma^2_{\mathsf{sufl}}(D_{2n})$	0.65391	1.26514	2.02532	2.29851	
	100		0.62294	1.32720	2.49557	3.98183	
	$\to\infty$		0.61983	1.33443	2.55590	4.26667	
5	10	$\mu_{\mathsf{sufl}}(D_{2n})$	1.30795	1.64243	2.10626	2.76323	3.71230
	100		1.30034	1.64449	2.14401	2.90690	4.15187
	$\to\infty$		1.29955	1.64474	2.14823	2.92398	4.21053
	10	$\sigma^2_{\mathsf{sufl}}(D_{2n})$	0.48929	0.91187	1.47111	2.06551	2.23756
	100		0.46442	0.92860	1.64749	2.70233	3.92519
	$\to\infty$		0.46189	0.93057	1.66828	2.78718	4.21053

$$C_{1,|\mathcal{R}|,|\mathcal{R}|} > C_{1,|\mathcal{R}|+1,|\mathcal{R}|+1}, \tag{4.12}$$

$$C_{1,|\mathcal{R}|,|T_{[}|} < C_{1,|\mathcal{R}|,|T_{[}|+1}. \tag{4.13}$$

TABLE 3

Exact and asymptotical values for $\mu_{\mathsf{sufl}}(D_{2n})$ *and* $\sigma^2_{\mathsf{sufl}}(D_{2n})$ *for relations with* $|\mathcal{R}| \in \{10, 20, 50, 100\}$ *and* $|T_{\lfloor}| \in \{1, \lceil \frac{|\mathcal{R}|}{4} \rceil, \frac{|\mathcal{R}|}{2}, \lceil \frac{3|\mathcal{R}|}{4} \rceil, |\mathcal{R}|\}$.

$\lvert\mathcal{R}\rvert \downarrow$	$n \downarrow$	$\lvert T_{\lfloor}\rvert \rightarrow$	1	$\lceil \frac{\lvert\mathcal{R}\rvert}{4} \rceil$	$\frac{\lvert\mathcal{R}\rvert}{2}$	$\lceil \frac{3\lvert\mathcal{R}\rvert}{4} \rceil$	$\lvert\mathcal{R}\rvert$
10	10	$\mu_{\mathsf{sufl}}(D_{2n})$	1.14058	1.41839	1.79957	2.68132	3.60298
	100		1.13684	1.41946	1.82102	2.83122	4.04378
	$\rightarrow \infty$		1.13644	1.41957	1.82336	2.84900	4.10256
	10	$\sigma^2_{\mathsf{sufl}}(D_{2n})$	0.21381	0.54447	1.01347	1.88943	2.12362
	100		0.20198	0.55127	1.10638	2.53566	3.81660
	$\rightarrow \infty$		0.20075	0.55203	1.11687	2.62173	4.10256
20	10	$\mu_{\mathsf{sufl}}(D_{2n})$	1.06728	1.32106	1.77318	2.46325	3.55073
	100		1.06545	1.32250	1.79762	2.57895	3.99182
	$\rightarrow \infty$		1.06526	1.32266	1.80028	2.59241	4.05063
	10	$\sigma^2_{\mathsf{sufl}}(D_{2n})$	0.09993	0.38904	0.93847	1.66690	2.07031
	100		0.09427	0.39533	1.03942	2.16155	3.76452
	$\rightarrow \infty$		0.09368	0.39601	1.05080	2.22487	4.05063
50	10	$\mu_{\mathsf{sufl}}(D_{2n})$	1.02623	1.32422	1.75772	2.47605	3.52012
	100		1.02551	1.32747	1.78386	2.60006	3.96128
	$\rightarrow \infty$		1.02543	1.32782	1.78671	2.61453	4.02010
	10	$\sigma^2_{\mathsf{sufl}}(D_{2n})$	0.03839	0.37172	0.89539	1.64838	2.03940
	100		0.03620	0.38376	1.00071	2.17339	3.73395
	$\rightarrow \infty$		0.03597	0.38505	1.01257	2.24113	4.02010
100	10	$\mu_{\mathsf{sufl}}(D_{2n})$	1.01300	1.30572	1.75262	2.43480	3.51003
	100		1.01264	1.30904	1.77932	2.55271	3.95120
	$\rightarrow \infty$		1.01261	1.30940	1.78223	2.56642	4.01003
	10	$\sigma^2_{\mathsf{sufl}}(D_{2n})$	0.01894	0.34303	0.88134	1.60367	2.02927
	100		0.01785	0.35496	0.98804	2.10131	3.72387
	$\rightarrow \infty$		0.01774	0.35623	1.00006	2.16505	4.01003

Altogether, we obtain the following values for $C_{1,|\mathcal{R}|,|T_{\lfloor}|}$ given in Table 1. Here, $\uparrow$, $\nwarrow$, and $\rightarrow$ stand for strictly increasing sequences. Obviously, $\uparrow$, $\nwarrow$, and $\rightarrow$ correspond to (4.11), (4.12), and (4.13), respectively.

Now, we take a look at two plots of $C_{1,|\mathcal{R}|,|T_{\lfloor}|}$ for $1 \leq |T_{\lfloor}| \leq |\mathcal{R}| \leq 10$ from two different points of view, analytically continued to $\mathbb{R}$.

By these plots, we get a better idea of the behavior of $\mu_{\mathsf{sufl}}(D_{2n})$ for different combinations of the two parameters $|\mathcal{R}|$ and $|T_{\lfloor}|$. In the left plot of Figure 10, we see clearly the inequations (4.11) and (4.12); in the right plot we recognize (4.13).

Remark 4.2. In [Li96] the lexicographical generation of the *Dyck language with t types of brackets* D^t [Ha78, p. 313] has been analyzed. The results are presented in [Ke98]. Following our definitions, $D^t = D$ with the relation $\mathcal{R} = \{([_1,]_1), ([_2,]_2), \ldots, ([_t,]_t)\}$, and thus $D^t_{2n} = D_{2n}$. Obviously, D^t_{2n} is simply generated (see Remark 3.1). The mean value and the variance of the random variable $X_{\mathsf{sufl}}(D^t_{2n})$ describing the number of symbols to be changed while generating the successor of a word were found to be

$$\mu_{\mathsf{sufl}}(D^t_{2n}) \sim \frac{16t}{4t-1}, \quad n \to \infty, \quad \text{and} \quad \sigma^2_{\mathsf{sufl}}(D^t_{2n}) \sim \frac{16t}{4t-1}, \quad n \to \infty.$$

For $\mathcal{R} = \{([_1,]_1), \ldots, ([_t,]_t)\}$, we obtain $|\mathcal{R}| = |T_{\lfloor}| = t$ and so by Corollary 4.3 $\mu_{\mathsf{sufl}}(D_{2n}) \sim \frac{16t}{4t-1}$, $n \to \infty$, and $\sigma^2_{\mathsf{sufl}}(D_{2n}) \sim \frac{16t}{4t-1}$, $n \to \infty$, evidently.

In Tables 2 and 3, we give some exact and asymptotical values for $\mu_{\mathsf{sufl}}(D_{2n})$ and $\sigma^2_{\mathsf{sufl}}(D_{2n})$ for various relations $\mathcal{R}$.

5. Lexicographical generation of labeled trees. It is well known that the normal Dyck language (with only one pair of brackets) is a coding for unlabeled trees such as ordered trees (with arbitrary degree of the nodes) and extended ordered binary trees (each node is a leaf or has exactly two sons); see, e.g., [Za80]. Other representations of the shape of trees are the integer sequences: level-representation, leaf-representation, and leaf-level-representation (see, e.g., [Ke98]). In contrast to the aforementioned representation of trees, the generalized Dyck language is a handy coding not only for the shape but also for the labels at the nodes and/or edges of several classes of trees. Some classes of labeled trees, coded by the generalized Dyck language, are discussed in [Li00]. There, a tree is coded in one Dyck word and not in two independent parts (one for the shape of the tree and another one for the labels). Nevertheless, the structure of the tree and its labels can be determined in a very simple way. Thus, with the algorithm presented here, we are able to generate classes of labeled trees lexicographically.

6. Concluding remarks. In this paper, we have presented an algorithm that generates all words of a generalized Dyck language lexicographically. The Dyck language is defined by a relation $\mathcal{R}$ which describes the pairs of brackets that can be used. We introduced a function that computes from one Dyck word the next one according to the lexicographical order. For that purpose, no knowledge about the words generated before is required.

Since the generalized Dyck language is a coding for several classes of labeled trees, we can generate these classes of trees lexicographically with the algorithm presented here.

Further, we found a condition for the generalized Dyck language to be simply generated, which means that for every word it is possible to compute its successor by reading only the suffix to be changed. We saw that this condition depends on the relation $\mathcal{R}$ only and not on the length of the words. We introduced an algorithm that computes whether the Dyck language—implied by the relation $\mathcal{R}$—is simply generated or not. The running time of that algorithm depends on the relation only, so it has a constant amount of time with respect to the length of the words.

Following a general approach to the lexicographical generation of all words of a formal language [Ke98], we computed the sth moments, $s \geq 1$, of the random variable describing the length of the suffix of a word to be changed. In particular, we pointed out the mean value and the variance of the number of symbols to be changed in order to generate the successor of a Dyck word.

Acknowledgment. The author would like to thank Prof. Rainer Kemp for his idea of how to compute the sth moments of the random variable describing the length of the suffix to be changed.

REFERENCES

[BLP98] E. Barcucci, A. Del Lungo, E. Pergola, and R. Pinzani, *A methodology for plane tree enumeration,* Discrete Math., 180 (1998), pp. 45–64.

[CRS71] L. Carlitz, D. P. Roselle, and R. A. Scoville, *Some remarks on ballot-type sequences of positive integers,* J. Combin. Theory Ser. A, 11 (1971), pp. 258–271.

[GKP94] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics,* 2nd ed., Addison–Wesley, Reading, MA, 1994.

[GK82] D. H. Greene and D. E. Knuth, *Mathematics for the Analysis of Algorithms,* 2nd ed., Birkhäuser Boston, Boston, 1982.

[Ha78] M. A. Harrison, *Introduction to Formal Language Theory*, Addison–Wesley, Reading, MA, 1978.

[Ke84] R. Kemp, *Fundamentals of the Average Case Analysis of Particular Algorithms*, Wiley–Teubner, Ser. Comput. Sci., John Wiley and Sons, Chichester, UK, 1984.

[Ke96] R. Kemp, *On the average minimal prefix-length of the generalized semi-Dycklanguage,* RAIRO Inform. Théor. Appl., 30 (1996), pp. 545–561.

[Ke98] R. Kemp, *Generating words lexicographically: An average-case analysis,* Acta Inform., 35 (1998), pp. 17–89.

[Li96] J. Liebehenschel, *Lexikographische Erzeugung der Dycksprachen mit mehreren Klammertypen,* Diplomarbeit, Fachbereich Informatik, Johann Wolfgang Goethe-Universität, Frankfurt am Main, Germany, 1996.

[Li98] J. Liebehenschel, *Lexicographical Generation of a Generalized Dyck Language,* Technical Report, Interner Bericht 5/98, Fachbereich Informatik, Johann Wolfgang Goethe-Universität, Frankfurt am Main, Germany.

[Li00] J. Liebehenschel, *Ranking and unranking of a generalized Dyck language and the application to the generation of random trees,* Sém. Lothar. Combin., 43 (2000), B43d.

[NW78] A. Nijenhuis and H. S. Wilf, *Combinatorial Algorithms for Computers and Calculators,* 2nd ed., Academic Press, New York, 1978.

[Wi77] H. S. Wilf, *A unified setting for sequencing, ranking and selection algorithms for combinatorial objects,* Adv. Math., 24 (1977), pp. 281–291.

[Za80] S. Zaks, *Lexicographic generation of ordered trees,* Theoret. Comput. Sci., 10 (1980), pp. 63–82.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 904–915

PARTIAL MATCH QUERIES IN RANDOM QUADTREES*

HUA-HUAI CHERN† AND HSIEN-KUEI HWANG‡

Abstract. We propose a simple direct approach for computing the expected cost of random partial match queries in random quadtrees. This approach gives not only an explicit expression for the leading constant in the asymptotic approximation of the expected cost but also more terms in the asymptotic expansion if desired.

Key words. quadtrees, partial match queries, binomial transform, Mellin transform, Euler transform, Rice's integral

AMS subject classifications. Primary, 68W40; Secondary, 68P05, 68P10

DOI. 10.1137/S0097539702412131

1. Introduction.

Quadtrees, originally proposed by Finkel and Bentley [2], represent one of the simplest, most prototypical, and most studied data structures for multidimensional data; see [13, 14]. We are concerned in this paper with the average-case analysis of random partial match queries in random quadtrees, a problem originally analyzed in detail by Flajolet et al. [3] and then extended by several authors.

The probabilistic model is as follows. First, a sequence of n points is generated uniformly and independently in $[0,1]^d$, where $d \geq 2$. From this sequence, we construct the (d-dimensional) quadtree and call it the *random quadtree*. (A quadtree of a sequence of points is constructed by placing the first element at the root, which splits the space $[0,1]^d$ into 2^d quadrants; the remaining points are compared to the root and are directed to each quadrant depending on the location of each point; points falling in each quadrant are constructed recursively as quadtrees.) Then we generate s independent random variables, with the same Uniform$[0,1]$ distribution, which correspond to the specified coordinates of the partial match query $\mathtt{q}$, where $1 \leq s < d$; the remaining $d-s$ coordinates are "wild-cards." We then perform the range search of this query $\mathtt{q}$ in the random tree and denote the expected number of nodes visited by $Q_n = Q_n^{(d,s)}$. Note that, by symmetry, only the number of specified coordinates matters; the order or positions of the specifications are immaterial.

Flajolet et al. [3] showed that Q_n can be computed recursively as follows (see Lemma 1 for a self-contained proof): $Q_0 = 0$ and for $n \geq 1$

$$Q_n = 1 + 2^d \sum_{1 \leq k < n} \pi_{n,k} Q_k, \tag{1.1}$$

*Received by the editors July 30, 2002; accepted for publication (in revised form) December 12, 2002; published electronically June 10, 2003.
http://www.siam.org/journals/sicomp/32-4/41213.html

†Department of Computer Science, National Taiwan Ocean University, 2 Pei Ning Road, Keelung 20224, Taiwan (felix@cs.ntou.edu.tw). The work of this author was partially supported by the National Science Council of the Republic of China under grant NSC-90-2115-M-133-001. The work of this author was performed while he was at Taipei Municipal Teachers College.

‡Institute of Statistical Science, Academia Sinica, Taipei 115, Taiwan (hkhwang@stat.sinica.edu.tw). The work of this author was partially supported by the National Science Council of the Republic of China under grant NSC-90-2118-M-001-034.

where

(1.2)
$$\pi_{n,k} = \frac{1}{n(n+1)} \sum_{k \le \ell_{d-1} \le \cdots \le \ell_1 < n} \frac{1}{(\ell_1+2)\cdots(\ell_{s-1}+2)} \times \frac{1}{(\ell_{s+1}+1)\cdots(\ell_{d-1}+1)}.$$

From this recurrence, they studied the differential system satisfied by the generating function $Q(z) := \sum_n Q_n z^n$ and then showed that

$$Q_n \sim h_{d,s}\, n^{\alpha-1}$$

for some constant $h_{d,s}$, where $1 < \alpha < 2$ solves the indicial equation $\phi(z) = 0$, with

$$\phi(z) := z^{d-s}(z+1)^s - 2^d.$$

In particular, when $d = 2$, $\alpha = (\sqrt{17}-1)/2$ and

$$h_{2,1} = \frac{\Gamma(2\alpha)}{2\Gamma^3(\alpha)}.$$

The calculation of the leading constants $h_{d,s}$ remains open for other values of (d, s).

The aim of this paper is to derive more precise asymptotic approximations for Q_n with an explicit expression for $h_{d,s}$ for all cases of (d, s).

THEOREM 1. *The expected number of nodes visited by performing the range search of a random partial match query in a random quadtree of n nodes satisfies*

(1.3)
$$Q_n = h_{d,s} n^{\alpha-1} + O(1 + n^{\Re(\alpha_2)-1}),$$

where

$$h_{d,s} := \frac{1}{(2^{d-s}-1)\Gamma(\alpha)^{d-s}\Gamma(\alpha+1)^s} \prod_{2 \le j \le d} \frac{\Gamma(\alpha-\alpha_j)}{\Gamma(1-\alpha_j)}$$

for $1 \le s < d$ and $d \ge 2$, Γ is the Gamma function, and the α_j's are zeros of the polynomial $\phi(z)$:

$$\alpha = \alpha_1 > \Re(\alpha_2) \ge \cdots \ge \Re(\alpha_d).$$

See Table 1.1 for numeric values of α and $h_{d,s}$ for $d \le 6$.

Our approach indeed gives a closed-form expression for Q_n.

PROPOSITION 1. *For $n \ge 1$, Q_n satisfies*

(1.4)
$$Q_n = \sum_{1 \le k \le n} \binom{n}{k} (-1)^{k+1} \frac{2^s (2-\alpha_1)_{k-1} \cdots (2-\alpha_d)_{k-1}}{k!^{d-s}(k+1)!^s},$$

where $(x)_k := x(x+1)\cdots(x+k-1)$.

Once the exact formula (1.4) is known, the proof of (1.3) uses the integral representation (or the Rice integral) for (1.4); the remaining analysis is more or less standard (see Flajolet and Sedgewick [7]). The main step in deriving (1.3) is thus the proof of (1.4). We propose a very simple elementary proof for (1.4) which uses only the binomial transform of Q_n and a differencing argument. Such a proof was

TABLE 1.1
Numeric values of $\alpha = \alpha_{d,s}$ and $h_{d,s}$ for $2 \le d \le 6$; observe that $\alpha_{2d,2s} = \alpha_{d,s}$.

d	s	α	$h_{d,s}$
2	1	1.5615 5281	1.5950 9909
3	1	1.7161 8865	1.1094 7781
3	2	1.3948 5867	1.7203 8918
4	1	1.7899 5097	0.9745 2299
4	2	1.5615 5281	1.1052 1638
4	3	1.3055 5316	1.7360 4197
5	1	1.8332 3029	0.9165 1089
5	2	1.6555 6266	0.9218 4703
5	3	1.4632 3881	1.0739 3189
5	4	1.2495 6226	1.7222 1994
6	1	1.8617 0559	0.8862 3369
6	2	1.7161 8865	0.8391 1604
6	3	1.5615 5281	0.8690 3393
6	4	1.3948 5867	1.0385 7434
6	5	1.2110 6870	1.7007 4787

originally motivated by a different approach based on the generating function, differential equations, and Euler transforms (cf. Flajolet et al. [4]). We will sketch this approach as well as an intuitive use of the Mellin–Barnes integral (see [12]) to give more insight into the problem. Although these approaches are essentially the same for Q_n, the elementary approach is computationally and technically much simpler; we will see later that it is also more general.

For other results on partial or exact match queries in random quadtrees, see [1, 5, 10, 11]. On the other hand, explicit characterization of the leading constants of partial match queries in random k-d trees (k being the dimension) is very different from that in quadtrees and is much harder (see [6]); this problem is treated elsewhere.

2. Splitting probabilities and recurrence. We first prove the recurrence (1.1), which is basic to our analysis.

LEMMA 1. *The expected cost Q_n satisfies the recurrence* (1.1) *with $\pi_{n,k}$ given by* (1.2).

Proof. Let $p_{n,k}$ denote the probability that the query is conducted in the first subtree (namely, its root is visited) whose size is k. By our independence assumptions and by symmetry, we may assume that the first s coordinates of the query are specified. It follows that Q_n satisfies (1.1) with

$$\begin{aligned}\pi_{n,k} &= 2^{-s} p_{n,k} \\ &= \binom{n-1}{k} \int_{(0,1)^d} x_1 \cdots x_s (x_1 \cdots x_d)^k (1 - x_1 \cdots x_d)^{n-1-k} \mathbf{dx}, \end{aligned} \tag{2.1}$$

where $\mathbf{dx} := \mathrm{d}x_1 \cdots \mathrm{d}x_d$. By expanding the factor $1 - x_1 \cdots x_d$ as

$$1 - x_1 \cdots x_d = 1 - x_1 + (1 - x_2)x_1 + \cdots + (1 - x_d)x_{d-1} \cdots x_1,$$

we then have (with the convention that $j_{d+1} = 0$)

$$\begin{aligned}\pi_{n,k} &= \binom{n-1}{k} \sum_{j_1+\cdots+j_d=n-1-k} \binom{n-1-k}{j_1,\ldots,j_d} \\ &\quad \times \int_{(0,1)^d} \left(\prod_{1\le i\le s} x_i^{k+1+j_{i+1}+\cdots+j_d} (1-x_i)^{j_i} \right) \\ &\quad \times \left(\prod_{s<i\le d} x_i^{k+j_{i+1}+\cdots+j_d} (1-x_i)^{j_i} \right) \mathbf{dx} \\ &= \sum_{j_1+\cdots+j_d=n-1-k} \frac{(n-1)!}{k!} \left(\prod_{1\le i\le s} \frac{(k+1+j_{i+1}+\cdots+j_d)!}{(k+2+j_i+\cdots+j_d)!} \right) \\ &\quad \times \left(\prod_{s<i\le d} \frac{(k+j_{i+1}+\cdots+j_d)!}{(k+1+j_i+\cdots+j_d)!} \right) \\ &= \frac{1}{n(n+1)} \sum_{j_1+\cdots+j_d=n-1-k} \frac{1}{(k+2+j_2+\cdots+j_d)\cdots(k+2+j_s+\cdots+j_d)} \\ &\quad \times \frac{1}{(k+1+j_{s+1}+\cdots+j_d)\cdots(k+1+j_d)},\end{aligned}$$

which is easily seen to be identical to (1.2). By symmetry, we then obtain (1.1). □

COROLLARY 1. *The sequence $\pi_{n,k}$ satisfies*

$$\pi_{n,k} = \sum_{k\le j<n} \binom{n-1}{j} \binom{j}{k} (-1)^{j+k} (j+1)^{-d+s} (j+2)^{-s}. \tag{2.2}$$

Proof. Expand the factor $(1-x_1\cdots x_d)^{n-1-k}$ in (2.1), and then evaluate the integral term by term. □

Note that, by (1.2) and (2.2), we have the identities

$$\begin{aligned}\sum_{k\le j<n} \binom{n-1}{j} \binom{j}{k} (-1)^{j+k} (j+1)^{-1} (j+2)^{-1} &= \frac{n-k}{n(n+1)}, \\ \sum_{k\le j<n} \binom{n-1}{j} \binom{j}{k} (-1)^{j+k} (j+1)^{-2} (j+2)^{-1} &= \frac{H_n - H_k}{n} - \frac{n-k}{n(n+1)}, \\ \sum_{k\le j<n} \binom{n-1}{j} \binom{j}{k} (-1)^{j+k} (j+1)^{-1} (j+2)^{-2} &= \frac{(n+2)(n-k)}{n(n+1)^2} \\ &\quad - \frac{(k+1)(H_n - H_k)}{n(n+1)},\end{aligned}$$

corresponding, respectively, to $(d,s) = (2,1), (3,1)$, and $(3,2)$, where $H_k := \sum_{1\le j\le k} 1/j$. More identities can be derived by considering higher values of d.

3. Binomial transform. The crucial step in proving (1.4) is to consider the binomial transform

$$Q_n^\star := \sum_{1\le k\le n} \binom{n}{k} (-1)^k Q_k \qquad (n \ge 1),$$

with $Q_0^\star := 0$.

LEMMA 2. *The sequence $Q_n^\star$ satisfies the first-order recurrence*

$$Q_n^\star - Q_{n-1}^\star = -\frac{2^d}{n^{d-s}(n+1)^s}\, Q_{n-1}^\star \qquad (n \geq 2), \tag{3.1}$$

with $Q_1^\star = -1$.

Proof. By (1.1),

$$\begin{aligned} Q_n^\star - Q_{n-1}^\star &= 2^d \sum_{1\leq m\leq n} \binom{n-1}{m-1} (-1)^m \sum_{1\leq k<m} \pi_{m,k} Q_k \\ &= -2^d \sum_{1\leq k<n} Q_k \sum_{k\leq m<n} \binom{n-1}{m} (-1)^m \pi_{m+1,k}. \end{aligned}$$

Now the inner sum can be simplified by using (2.1) as follows:

$$\begin{aligned} &\sum_{k\leq m<n} \binom{n-1}{m} (-1)^m \pi_{m+1,k} \\ &\quad = \sum_{k\leq m<n} \binom{n-1}{m} (-1)^m \binom{m}{k} \int_{(0,1)^d} x_1\cdots x_s (x_1\cdots x_d)^k (1-x_1\cdots x_d)^{m-k} \mathbf{dx} \\ &\quad = \binom{n-1}{k} (-1)^k \sum_{0\leq m\leq n-1-k} \binom{n-1-k}{m} (-1)^m \\ &\qquad \times \int_{(0,1)^d} x_1\cdots x_s (x_1\cdots x_d)^k (1-x_1\cdots x_d)^m \mathbf{dx} \\ &\quad = \binom{n-1}{k} (-1)^k \int_{(0,1)^d} x_1\cdots x_s (x_1\cdots x_d)^{n-1} \mathbf{dx} \\ &\quad = \binom{n-1}{k} (-1)^k n^{-d+s} (n+1)^{-s}, \end{aligned}$$

which holds for $n \geq 2$. □

Proof of (1.4). From Lemma 2, it follows that, for $n \geq 2$,

$$Q_n^\star = -\prod_{2\leq j\leq n} \left(1 - \frac{2^d}{(j+1)^s j^{d-s}}\right) = -\prod_{2\leq j\leq n} \frac{\phi(j)}{(j+1)^s j^{d-s}},$$

which leads to (1.4) by the relation

$$Q_n = \sum_{1\leq j\leq n} \binom{n}{j} (-1)^j Q_j^\star.$$

This proves (1.4). □

4. Generating function, differential equation, and Euler transform. While binomial transform is shown to be the "Eureka!" to our elementary proof of (1.4), it leaves open how such a transform was linked and applied to our problem. In this section, we give an alternative approach to deriving the simple recurrence (3.1),

relying on the generating function $Q(z) := \sum_n Q_n z^n$ and the Euler transform (see [4])

$$Q^\star(z) = \frac{1}{1-z}\, Q\left(\frac{z}{z-1}\right). \tag{4.1}$$

Note that the coefficients f_n of a function $f(z)$ and those $f_n^\star$ of its Euler transform $f^\star(z)$ satisfy the inversion pair

$$\begin{cases} f_n = \displaystyle\sum_{0\le k\le n} \binom{n}{k} (-1)^k f_k^\star, \\ f_n^\star = \displaystyle\sum_{0\le k\le n} \binom{n}{k} (-1)^k f_k. \end{cases} \tag{4.2}$$

We start by substituting the expression (2.2) into the sum $\sum_{1\le k<n} \pi_{n,k} Q_k$, which gives

$$\begin{aligned} \sum_{1\le k<n} \pi_{n,k} Q_k &= \sum_{1\le k<n} \sum_{k\le j<n} \binom{n-1}{j}\binom{j}{k} (-1)^{j+k} (j+1)^{-d+s} (j+2)^{-s} Q_k \\ &= \sum_{1\le j<n} \binom{n-1}{j} (-1)^j (j+1)^{-d+s} (j+2)^{-s} Q_j^\star. \end{aligned}$$

In terms of generating functions, the recurrence (1.1) then translates into (in view of (4.2))

$$Q(z) = \frac{z}{1-z} + 2^d \frac{z}{1-z} S\left(\frac{z}{z-1}\right), \tag{4.3}$$

where

$$S(w) = \sum_{n\ge 1} (n+1)^{-d+s} (n+2)^{-s} Q_n^\star w^n.$$

Note that $S(w)$ satisfies

$$\vartheta^{d-s}\left(w^{-1}\vartheta^s\left(w^2 S(w)\right)\right) = w \sum_{n\ge 1} Q_n^\star w^n = \frac{w}{1-w} Q\left(\frac{w}{w-1}\right), \tag{4.4}$$

where $\vartheta := w(\mathrm{d}/\mathrm{d}w)$.

Thus it is natural to consider the Euler transform. Taking $z = w/(w-1)$ and then multiplying (4.3) by $-w$, we obtain

$$w(w-1)Q^\star(w) = w^2 + 2^d w^2 S(w).$$

From (4.4), it follows that

$$\vartheta^{d-s}\left(w^{-1}\vartheta^s\left(w(w-1)Q^\star(w)\right)\right) - 2^d w Q^\star(w) = 2^s w. \tag{4.5}$$

Taking the coefficients of w^n on both sides yields (3.1).

5. Differential equation and Mellin–Barnes integrals. More light can be shed on "why the Euler transform really helps" by looking at yet another approach, this one based on the differential equation derived by using the original form (1.2) for $\pi_{n,k}$ used in [3].

The proof starts by showing that the generating function $Q(z)$ satisfies, by (1.1) and (1.2), the integro-differential equation

$$\left(z(1-z)\frac{\mathrm{d}}{\mathrm{d}z}-\mathbf{1}\right)\frac{z}{1-z}\,Q(z)=\frac{z^2}{(1-z)^2}+2^d\mathbf{J}^s\frac{z}{1-z}\mathbf{J}^{d-s-1}(zQ(z)), \tag{5.1}$$

where $\mathbf{1}$ is the identity operator and

$$\mathbf{J}f(z):=\int_0^z\frac{f(t)}{t(1-t)}\,\mathrm{d}t.$$

Introduce now the differential operator $\mathcal{J}:=z(1-z)\,\mathrm{d}/\,\mathrm{d}z$. Then (5.1) can be shown to be of the form

$$\mathcal{J}^{d-s}\frac{1-z}{z}\mathcal{J}^s\left(\frac{z}{1-z}\,Q(z)-\frac{z^2}{(1-z)^2}\right)-2^dzQ(z)=0.$$

An inspiring way to see how to further proceed from this differential equation is to use the Mellin–Barnes integrals (see [12]) as follows. If a function is expressible formally as an integral of the form

$$f(z)=\frac{1}{2\pi i}\int_{\mathcal{C}}\tilde{f}(v)z^v(1-z)^{-v}\,\mathrm{d}v$$

for some contour $\mathcal{C}$ in the v-plane, then

$$\begin{aligned}\mathcal{J}^kf(z)&=\frac{1}{2\pi i}\int_{\mathcal{C}}\tilde{f}(v)\mathcal{J}^k\left(z^v(1-z)^{-v}\right)\,\mathrm{d}v\\&=\frac{1}{2\pi i}\int_{\mathcal{C}}\tilde{f}(v)v^kz^v(1-z)^{-v}\,\mathrm{d}v\qquad(k\ge1).\end{aligned}\tag{5.2}$$

Note that no such simplification is available if we use the usual Mellin–Barnes integrals

$$f(z)=\frac{1}{2\pi i}\int_{\mathcal{C}}\tilde{f}(v)(1-z)^{-v}\,\mathrm{d}v$$

or

$$f(z)=\frac{1}{2\pi i}\int_{\mathcal{C}}\tilde{f}(v)z^{-v}\,\mathrm{d}v.$$

The relation (5.2) suggests that it is natural to apply the Euler transform (4.1), and we obtain again (4.5).

6. Asymptotics of $Q(z)$. Since Q_n is essentially the nth difference of $Q_n^\star$, we have (see [7])

$$Q_n=\frac{1}{2\pi i}\int_{\sigma-i\infty}^{\sigma+i\infty}\frac{(-1)^nn!}{v(v-1)\cdots(v-n)}\,K(v+1)\,\mathrm{d}v, \tag{6.1}$$

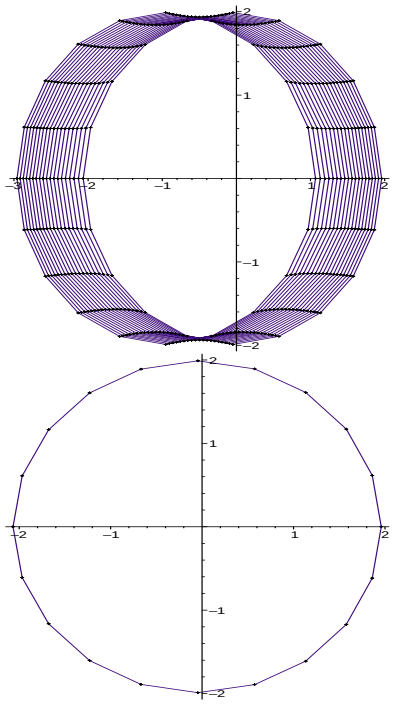


FIG. 6.1. *Distribution of the zeros of the polynomial* $z^{20-s}(z+1)^s = 2^{20}$ *for* $1 \le s \le 19$ *(top) and for* $s = 1$ *(bottom).*

where $\alpha_1 - 1 < \sigma < 1$ and

$$K(v) := \frac{2^s \Gamma(v-\alpha_1)\cdots\Gamma(v-\alpha_d)}{\Gamma(v)^{d-s}\Gamma(v+1)^s\Gamma(2-\alpha_1)\cdots\Gamma(2-\alpha_d)}.$$

We list some properties of the zeros α_j of $\phi(z)$ as follows; see Figure 6.1 for a plot of the zeros of $\phi(z)$ when $d = 20$ and $1 \le s \le 19$.

(i) *All zeros lie within the ring* $3/2 < |z + 1/2| < 5/2$. This follows by noting that

$$(|z| - 1/2)^d \le |z - 1/2|^{d-s}|z + 1/2|^s \le (|z| + 1/2)^d$$

and by using the bounds $|z| - 1/2 \ge 2$ and $|z| + 1/2 \le 2$.

(ii) *The zero with the largest real part is* α. To see this, assume that $\alpha_j = x + iy$ with $x \ge \alpha$ and $|y| > 0$; then we have

$$\begin{aligned} |\,(x+iy)^{d-s}\,(x+iy+1)^s| &= \left(x^2+y^2\right)^{(d-s)/2}\left((x+1)^2+y^2\right)^{s/2} \\ &> x^{d-s}(x+1)^s \ge \alpha^{d-s}(\alpha+1)^s = 2^d. \end{aligned}$$

(iii) *If* d *is odd, then* $\phi(z)$ *has only one real zero* α_1; *if* d *is even, then* $\phi(z)$ *has an additional real root lying between* -2 *and* -3.

(iv) *All zeros of* $\phi(z)$ *are simple.*

From these properties, it follows that the origin is a simple pole of the integrand in (6.1) (with residue $-(2^{d-s}-1)^{-1}$), and there are simple poles at $\alpha_j - 1 - \ell$ for $1 \le j \le d$ and $\ell \ge 0$. Other singularities are a pole of order $d - s$ at -1 and a pole at

$-\ell$ of order d for $\ell \geq 2$. In particular, the dominant singularity (the singularity with the largest real part in the half-plane $\Re(v) < 1$) is a simple pole at $v = \alpha - 1$. Also, by the asymptotic estimate

$$|\Gamma(\sigma + it)| = O(|t|^{\sigma-1/2} e^{-\pi|t|/2}) \qquad (|t| \to \infty)$$

for fixed σ, we can shift, by absolute convergence, the integration line in (6.1) to $\Re(v) = \sigma_1$, where

$$\max\{0, \Re(\alpha_2) - 1\} < \sigma_1 < \alpha - 1,$$

and compute the residue encountered at $v = \alpha - 1$, which gives

$$Q_n = \frac{(-1)^n n!}{(\alpha-1)\cdots(\alpha-n-1)} \lim_{v\to\alpha} (v-\alpha)K(v) + E_n,$$

where

$$E_n := \frac{1}{2\pi i} \int_{\sigma_1 - i\infty}^{\sigma_1 + i\infty} \frac{(-1)^n n!}{v(v-1)\cdots(v-n)} K(v+1)\, \mathrm{d}v.$$

Observe that

$$\begin{aligned}
&\frac{(-1)^n n!}{(\alpha-1)\cdots(\alpha-n-1)} \lim_{v\to\alpha} (v-\alpha)K(v) \\
&\quad = \frac{2^s \Gamma(n+1)}{(\alpha-1)\Gamma(n+2-\alpha)\Gamma(\alpha)^{d-s}\Gamma(\alpha+1)^s} \prod_{2\leq j\leq d} \frac{\Gamma(\alpha-\alpha_j)}{\Gamma(2-\alpha_j)} \\
&\quad = h_{d,s} \frac{\Gamma(n+1)}{\Gamma(n+2-\alpha)},
\end{aligned}$$

where we used the identity $\phi(1) = (1-\alpha_1)\cdots(1-\alpha_d) = 2^s - 2^d$.

By applying the asymptotic approximation

$$\frac{\Gamma(n+1)}{\Gamma(n+2-\alpha)} = n^{\alpha-1}\left(1 + O(n^{-1})\right),$$

we deduce that

$$Q_n = h_{d,s} n^{\alpha-1}\left(1 + O(n^{-1})\right) + E_n.$$

By a similar argument, we have

$$E_n = O(1 + n^{\Re(\alpha_2)-1}).$$

This proves (1.3). □

Remarks. (i) If $\Re(\alpha_2) < 1$, then

$$Q_n = h_{d,s} n^{\alpha-1} - \frac{1}{2^{d-s}-1} + O(n^{\alpha-2} + n^{\Re(\alpha_2)-1}). \tag{6.2}$$

In particular, $\Re(\alpha_2) < 1$ for all $d \leq 6$ and $1 \leq s < d$. See Figure 6.2 for a plot of $(Q_n + (2^{d-s}-1)^{-1})/n^{\alpha-1}$ when $d = 3$.

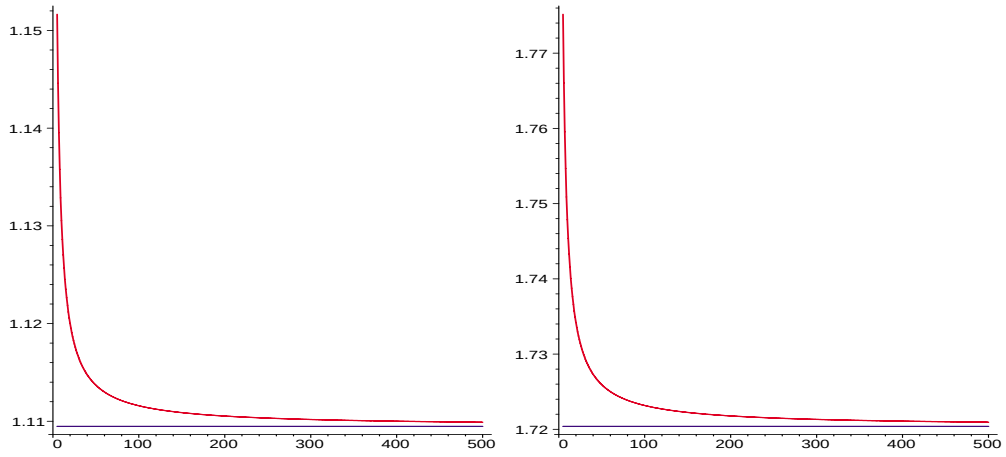


FIG. 6.2. *Convergence of* $(Q_n + (2^{d-s}-1)^{-1})/n^{\alpha-1}$ *to* $h_{d,s}$ *for* $(d,s) = (3,1)$ *(left) and* $(3,2)$ *(right), where* $n = 5, \dots, 500$.

(ii) If $\Re(\alpha_2) = \Re(\alpha_3) > 1$, then

$$Q_n = h_{d,s} n^{\alpha-1} + 2\Re(Y_{d,s}(\alpha_2) n^{i\Im(\alpha_2)}) n^{\Re(\alpha_2)-1} + O(n^{\alpha-2} + 1 + n^{\Re(\alpha_4)-1}),$$

where

$$Y_{d,s}(z) = \frac{2^s}{(z-1)\Gamma(z)^{d-s}\Gamma(z+1)^s} \prod_{\alpha_j \neq z} \frac{\Gamma(z-\alpha_j)}{\Gamma(2-\alpha_j)}.$$

Note that $h_{d,s} = Y_{d,s}(\alpha)$. More terms can be obtained by refining the same analysis.

(iii) By (1.4), the generating function $Q(z)$ can be expressed in terms of the generalized hypergeometric function (see [8]) as

$$Q(z) = \frac{z}{(z-1)^2}\, {}_{d+1}F_d\left(\begin{matrix} 2-\alpha_1, \dots, 2-\alpha_d, 1 \\ \underbrace{3, \dots, 3}_{s \text{ times}}, \underbrace{2, \dots, 2}_{d-s \text{ times}} \end{matrix} \,\middle|\, \frac{z}{z-1} \right),$$

where

$${}_{d+1}F_d\left(\begin{matrix} a_1, \dots, a_{d+1} \\ b_1, \dots, b_d \end{matrix} \,\middle|\, z \right) := \sum_{n \geq 0} \frac{(a_1)_n \cdots (a_{d+1})_n}{(b_1)_n \cdots (b_d)_n} \cdot \frac{z^n}{n!}.$$

7. General toll functions. The approaches we used can also be applied to a more general recurrence of the form

$$f_n = t_n + 2^d \sum_{1 \leq k < n} \pi_{n,k} f_k, \qquad (n \geq 1), \tag{7.1}$$

where t_n is some given sequence, and we may assume that $f_0 = 0$. We obtain the following first-order recurrence:

$$f^\star_n - f^\star_{n-1} = t^\star_n - t^\star_{n-1} - \frac{2^d}{n^{d-s}(n+1)^s} f^\star_{n-1} \qquad (n \geq 1),$$

with $t_0^\star = t_0 := 0$. Solving this recurrence yields

$$f_n^\star = \sum_{1\le k\le n} \left(t_k^\star - t_{k-1}^\star\right) \prod_{k<j\le n} \left(1 - \frac{2^d}{j^{d-s}(j+1)^s}\right) \qquad (n \ge 1). \tag{7.2}$$

By the binomial inversion formula (4.2), we obtain

$$f_n = \sum_{1\le k\le n} (t_k^\star - t_{k-1}^\star) \sum_{k\le m\le n} \binom{n}{m} (-1)^m \prod_{k<j\le m} \left(1 - \frac{2^d}{j^{d-s}(j+1)^s}\right)$$

for $n \ge 1$, where $t_n^\star := \sum_{1\le k\le n} \binom{n}{k}(-1)^k t_k$.

From this explicit form, the asymptotics of f_n can be derived if those of t_n are known.

8. Random full-specified queries. Our elementary approach is also useful for simplifying the recurrence encountered in the analysis of full-specified queries in random quadtrees (see [4, 5]):

$$f_n = t_n + 2^d \sum_{1\le k<n} \varpi_{n,k} f_k \qquad (n \ge 1), \tag{8.1}$$

with $f_0 = t_0 := 0$, where

$$\varpi_{n,k} = \binom{n-1}{k} \int_{(0,1)^d} (x_1 \cdots x_d)^k (1 - x_1 \cdots x_d)^{n-1-k} \mathbf{dx}.$$

From this, we then easily derive the recurrence relation (see [4, 10])

$$f_n^\star - f_{n-1}^\star = t_n^\star - t_{n-1}^\star - \frac{2^d}{n^d} f_{n-1}^\star \qquad (n \ge 1).$$

More generally, if

$$f_n = t_n + \sum_{1\le k<n} \beta_{n,k} f_k \qquad (n \ge 1),$$

where the sequence $\beta_{n,k}$ satisfies

$$\sum_{k\le m<n} \binom{n-1}{m} (-1)^m \beta_{m+1,k} = -\binom{n-1}{k} (-1)^k \varepsilon_n$$

for some sequence ε_n independent of k, then the recurrence for $f_n^\star$ can be "linearized,"

$$f_n^\star - f_{n-1}^\star = t_n^\star - t_{n-1}^\star - \varepsilon_n f_{n-1}^\star,$$

and we obtain the same pattern as above. Note that by (4.2),

$$\beta_{n,k} = \sum_{k\le j<n} \binom{n-1}{j} \binom{j}{k} (-1)^{j+k+1} \varepsilon_{j+1};$$

cf. (2.2).

A trivial example is the quicksort recurrence for which $\beta_{n,k} = 2/n$ for $0 \le k < n$. Then $\varepsilon_n = 2/n$. For other examples for which the Euler transform is particularly helpful, see [9] and the references therein.

This consideration also shows the limitation of the approach via the generating function and differential equations, since ε_n may be very complicated in general, so that the associated generating functions may not be reduced to simple solvable forms.

REFERENCES

[1] L. Devroye and L. Laforest, *An analysis of random d-dimensional quad trees*, SIAM J. Comput., 19 (1990), pp. 821–832.

[2] R. A. Finkel and J. L. Bentley, *Quadtrees: A data structure for retrieval on composite keys*, Acta Inform., 4 (1974), pp. 1–9.

[3] P. Flajolet, G. Gonnet, C. Puech, and J. M. Robson, *Analytic variations on quadtrees*, Algorithmica, 10 (1993), pp. 473–500.

[4] P. Flajolet, G. Labelle, L. Laforest, and B. Salvy, *Hypergeometrics and the cost structure of quadtrees*, Random Structures Algorithms, 7 (1995), pp. 117–114.

[5] P. Flajolet and T. Lafforgue, *Search costs in quadtrees and singularity perturbation asymptotics*, Discrete Comput. Geom., 12 (1994), pp. 151–175.

[6] P. Flajolet and C. Puech, *Partial match retrieval of multidimensional data*, J. ACM, 33 (1986), pp. 371–407.

[7] P. Flajolet and R. Sedgewick, *Mellin transforms and asymptotics: Finite differences and Rice's integrals*, Theoret. Comput. Sci., 144 (1995), pp. 101–124.

[8] P. Henrici, *Applied and Computational Complex Analysis, Vol.* 1, John Wiley, New York, 1973.

[9] F. Hubalek, H.-K. Hwang, W. Lew, H. M. Mahmoud, and H. Prodinger, *A multivariate view of random bucket digital search trees*, J. Algorithms, 44 (2002), pp. 121–158.

[10] G. Labelle and L. Laforest, *Étude de constantes universelles pour les arborescences hyperquaternaires de recherche*, Discrete Math., 153 (1996), pp. 199–211.

[11] R. Neininger and L. Rüschendorf, *Limit laws for partial match queries in quadtrees*, Ann. Appl. Probab., 11 (2001), pp. 452–469.

[12] D. Kaminski and R. B. Paris, *Asymptotics and Mellin-Barnes Integrals*, Cambridge University Press, Cambridge, UK, 2001.

[13] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison–Wesley, Reading, MA, 1990.

[14] H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, Addison–Wesley, Reading, MA, 1990.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 916–934

THE HIDDEN SUBGROUP PROBLEM AND QUANTUM COMPUTATION USING GROUP REPRESENTATIONS*

SEAN HALLGREN†, ALEXANDER RUSSELL‡, AND AMNON TA-SHMA§

Abstract. The hidden subgroup problem is the foundation of many quantum algorithms. An efficient solution is known for the problem over abelian groups, employed by both Simon's algorithm and Shor's factoring and discrete log algorithms. The nonabelian case, however, remains open; an efficient solution would give rise to an efficient quantum algorithm for graph isomorphism. We fully analyze a natural generalization of the algorithm for the abelian case to the nonabelian case and show that the algorithm determines the normal core of a hidden subgroup: in particular, normal subgroups can be determined. We show, however, that this immediate generalization of the abelian algorithm does not efficiently solve graph isomorphism.

Key words. quantum computation, quantum algorithms, computational complexity, representation theory, finite groups

AMS subject classifications. 81P68, 68Q17

DOI. 10.1137/S009753970139450X

1. Introduction.

Peter Shor's seminal article [27] presented efficient quantum algorithms for computing integer factorizations and discrete logarithms, problems thought to be intractable for classical computation models. A primary ingredient in these algorithms is an efficient solution to the *hidden subgroup problem* for certain abelian groups; indeed computing discrete logarithms directly reduces to the hidden subgroup problem. Formally, the hidden subgroup problem is the following.

DEFINITION 1.1. Hidden subgroup problem (HSP). *Given an efficiently computable function $f : G \to S$, from a finite group G to a set S, that is constant on (left) cosets of some subgroup H and takes distinct values on distinct cosets, determine the subgroup H.*

The general paradigm, which gives rise to efficient quantum algorithms for this problem over abelian groups, is the following.

EXPERIMENT 1.1 (experiment for the abelian HSP).

1. *Prepare the state*

$$\frac{1}{\sqrt{|G|}} \sum_{g \in G} |g, f(g)\rangle$$

and measure the second register $f(g)$. As f takes distinct values on the left cosets of

*Received by the editors August 28, 2001; accepted for publication (in revised form) December 16, 2002; published electronically June 10, 2003. A preliminary version of this article appeared in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, Portland, OR, 2000, pp. 627–635. The bulk of this research was completed while the authors were at the University of California, Berkeley.

http://www.siam.org/journals/sicomp/32-4/39450.html

†Computer Science Department, Caltech, MC 256-80, Pasadena, CA 91125 (hallgren@cs.caltech.edu). This author was supported in part by an NSF Mathematical Sciences Postdoctoral Fellowship, an NDSEG fellowship, a GAANN fellowship, and NSF grant CCR-9800024.

‡Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269 (acr@cse.uconn.edu). This author was supported by NSF CAREER award CCR-0093065, NSF grant CCR-0220264, NSF grant EIA-0218443, NSF NYI grant CCR-9457799, and a David and Lucile Packard Fellowship for Science.

§Computer Science Division, Tel-Aviv University, Israel 69978 (amnon@post.tau.ac.il).

H, the resulting state is

$$\frac{1}{\sqrt{|H|}} \sum_{h \in H} |ch, f(ch)\rangle, \tag{1.1}$$

where c is an element of G selected uniformly at random.

2. *Compute the Fourier transform of the "coset" state* (1.1), *resulting in*

$$\sum_{\rho \in \hat{G}} \sqrt{\frac{1}{|G|}} \sqrt{\frac{1}{|H|}} \sum_{h \in H} \rho(ch) |\rho\rangle,$$

where $\hat{G}$ denotes the set of homomorphisms $\{\rho : G \to \mathbb{C}\}$.

3. *Measure the first register and observe a homomorphism ρ.*

A key fact about this procedure is that the resulting distribution over ρ is independent of the coset cH arising after the first stage (as the support of the first register in (1.1)). Thus, repetitions of this experiment result in the same distribution over $\hat{G}$. We note that by the principle of delayed measurement (see, e.g., [21]), measuring the second register in the first step can in fact be delayed until the end of the experiment.

It is well known that an efficient solution to the HSP for the symmetric group S_n gives, in particular, an efficient quantum algorithm for graph isomorphism. It is also known how to efficiently compute the Fourier transform over many nonabelian groups, most notably over S_n [2]. This article provides the first general understanding of the HSP over nonabelian groups: We study a natural generalization of Experiment 1.1 to nonabelian groups and explicitly describe the resulting measurement distribution. Specifically, we study the following experiment.

EXPERIMENT 1.2.

1. *Prepare the state $\sum_{g \in G} |g, f(g)\rangle$ and measure the second register $f(g)$. The resulting state is $\sum_{h \in H} |ch, f(ch)\rangle$, where c is an element of G selected uniformly at random. As above, this state is supported on a left coset cH of H.*

2. *Let $\hat{G}$ denote the set of irreducible representations of G and, for each $\rho \in \hat{G}$, fix a basis for the space on which ρ acts. Let d_ρ denote the dimension of ρ. Compute the Fourier transform of the "coset" state, resulting in*

$$\sum_{\rho \in \hat{G}} \sum_{i}^{d_\rho} \sum_{j}^{d_\rho} \sqrt{\frac{d_\rho}{|G|}} \sqrt{\frac{1}{|H|}} \left(\sum_{h \in H} \rho(ch) \right)_{i,j} |\rho, i, j\rangle.$$

3. *Measure the first register and observe a representation ρ.*

A brief discussion of the representation theory of finite groups and the associated Fourier transform appears in section 2. As before, we wish the resulting distribution to be *independent* of the actual coset cH (and so depend only on the subgroup H). This is guaranteed by measuring only the name of the representation ρ and leaving the matrix indices (the values i and j) unobserved. The question we study is whether this procedure retains enough information to determine H or, more precisely, whether $O(\log(|G|))$ samples of this distribution are enough to determine H with high probability. Our analysis of Experiment 1.2 depends on the following theorem, which describes the distribution resulting from the measurements in the above experiment.

THEOREM 1.2. *Let H be a subgroup of a group G and let ρ be an irreducible representation of G with dimension d_ρ. Let $R_H(\rho)$ denote the number of times that the trivial representation appears in ρ when decomposed as a representation of H, and*

let $I_H(\rho)$ denote the number of times that ρ appears in the permutation representation of G on the left cosets of H. Then the probability of measuring the representation ρ in Experiment 1.2 *when H is the hidden subgroup is*

$$\frac{d_\rho \cdot R_H(\rho) \cdot |H|}{|G|} = \frac{d_\rho \cdot I_H(\rho) \cdot |H|}{|G|}. \tag{1.2}$$

We first apply this to obtain the following positive result.

THEOREM 1.3. *Let H be an arbitrary subgroup of G, and let H^G be the largest subgroup of H that is normal in G. With high probability, H^G is uniquely determined by observing $m = O(\log |G|)$ independent trials of Experiment* 1.2 *when H is the hidden subgroup.*

When H is normal in G, $H = H^G$, so that this algorithm determines H with high probability. In fact, we shall see that if $\rho_1, \ldots, \rho_m$ are the representations sampled by m independent runs of Experiment 1.2 and m is sufficiently large, then $H^G = \cap_i \ker \rho_i$ with high probability. (Here $\ker \rho$ denotes the kernel of the representation ρ.)

Our reconstruction result applies to any normal subgroup H of any group G without reference to the specific way that the representations or the group elements are expressed. We proceed at this level of abstraction because there is no known canonical concise presentation for the representations (or, indeed, the elements) of a finite group G. In the same vein, there is no general method for computing the Fourier transform over an arbitrary group. Thus, while we cannot give a unified algorithm for computing the Fourier transform or a set of generators for a hidden subgroup, this does yield an algorithm for any group which admits (i) a succinct representation over which the Fourier transform can be computed, and (ii) an efficient algorithm for computing the intersection of a (polynomial-size) family of representation kernels. See section 6, where the above approach is applied to solve the HSP for Hamiltonian groups, where all subgroups are normal.

Note that it is known [8] that the HSP has polynomial (in $\log |G|$) *query* complexity for any subgroup, though the only known algorithm which achieves this uses an exponential number of quantum measurements and, hence, does not give rise to an efficient quantum algorithm for the HSP.

A corollary of Theorem 1.2 is that conjugate subgroups H_1 and H_2 (where $H_2 = gH_1g^{-1}$ for some $g \in G$) produce exactly the same distribution over ρ and hence cannot be distinguished by this process. In particular, the HSP cannot be solved by Experiment 1.2 for a group G with two distinct conjugate subgroups H_1, H_2; the symmetric group S_n is such a group.

In light of this, one may ask whether Experiment 1.2 can distinguish between a coset cH of a nontrivial subgroup H and a coset $cH_e = \{c\}$ of the trivial subgroup $H_e = \{e\}$, as even this would be enough for solving graph isomorphism. However, even for this weaker problem we show (in section 5) the following.

THEOREM 1.4. *For the symmetric group S_n, there is a subgroup H_n so that Experiment* 1.2 *does not distinguish (even information-theoretically) the case that the hidden subgroup is the trivial subgroup from the case that the hidden subgroup is H_n. (Specifically, the distributions induced on ρ in these two cases have exponentially small distance in total variation.)*

1.1. Related work. The HSP plays a central role in most known quantum algorithms. Simon's algorithm [28] implicitly involves distinguishing the trivial subgroup from an order 2 subgroup over the group $\mathbb{Z}_2^n$. Furthermore, he has shown that a classical probabilistic oracle machine would require exponentially many oracle queries

to successfully distinguish the two cases with probability greater than 1/2. Shor [27] then gave efficient algorithms for integer factorization and the discrete log problem. In addition to solving a special case of the HSP, he also solved specific cases where the size of the underlying group is not fully known. Other generalizations have been studied by Boneh and Lipton [3], focusing on cases when a periodic function is not fixed on a coset, and Hales and Hallgren [11, 12], who generalized the results for the case when the underlying abelian group is unknown.

The efficient algorithm for the abelian HSP using the Fourier transform is well known. Other methods have been applied to this same problem by Mosca and Ekert [19]. Related problems have been studied by Kitaev [17], who gave an algorithm using eigenvalue estimation for the abelian stabilizer problem, and Hallgren [13], who gave polynomial-time quantum algorithms for Pell's equation and the principal ideal problem.

As for computing the Fourier transform, Kitaev showed how to efficiently compute the Fourier transform over any abelian group. The fastest currently known (quantum) algorithm for computing the Fourier transform over abelian groups was given by Hales and Hallgren [12]. Shallow parallel circuits for approximating the Fourier transform have been given by Cleve and Watrous in [4]. Beals [2] showed how to efficiently compute the Fourier transform over the symmetric group S_n.

For general groups, Ettinger, Høyer, and Knill [8] have shown that the HSP has polynomial query complexity, giving an algorithm that makes an exponential number of measurements. On the other hand, if one considers arbitrary functions rather than those that arise from HSPs, Aaronson [1] shows that it is not possible to distinguish a 1-1 function from a 2-1 function, even with a quantum algorithm. Several specific nonabelian groups have been studied in the context of the HSP. Ettinger and Høyer [7] give a solution for the HSP over the (nonabelian) dihedral group D_n using polynomially many measurements and exponential (classical) time. Rötteler and Beth [24] and Püschel, Rötteler, and Beth [22] have shown similar results for other specific classes of nonabelian groups. Ivanyos, Mangniez, and Santha [16] have shown how to solve certain nonabelian HSP instances using a reduction to the abelian case.

Grigni et al. [10] independently showed that measuring the representation is not enough for graph isomorphism, and they give stronger negative results. They establish the same bounds even when the row of the representation (i.e., i in Experiment 1.2 above) is measured, and similar bounds if the column (j) is measured, under the assumption that random bases are selected for each representation. They also show that the problem can be solved when the intersection of the normalizers of all subgroups of G is large. Other impossibility results have been given by Ettinger and Høyer [6, 5], determining whether *any* measurement can distinguish certain subgroup states.

2. Representation theory background. To define the Fourier transform (over a general group) we require the basic elements of representation theory, defined briefly below. For complete accounts, consult the books of Serre [26] or Harris and Fulton [15]. Throughout, we let $\mathbb{I}_d$ denote the $d \times d$ identity matrix, dropping the subscript when it can be inferred from context.

Linear representations. A representation ρ of a finite group G is a homomorphism $\rho : G \to GL(V)$, where V is a (finite-dimensional) vector space over $\mathbb{C}$ and $\mathrm{GL}(V)$ denotes the group of invertible linear operators on V. Fixing a basis for V, each $\rho(g)$ may be realized as a $d \times d$ matrix over $\mathbb{C}$, where d is the dimension of V. As ρ is a homomorphism, for any $g, h \in G$, $\rho(gh) = \rho(g)\rho(h)$ (this second product being matrix multiplication). The *dimension* d_ρ of the

representation ρ is d, the dimension of V.

A representation provides a means for investigating a group by homomorphically mapping it into a family of matrices. With this realization, the group operation is matrix multiplication, and tools from linear algebra can be applied to study the group. We shall be concerned with complex-valued functions on a group G; the representations of the group are relevant to this study, as they give rise to the natural Fourier transform in this nonabelian setting.

If $\rho : G \to \mathrm{GL}(V)$ is a representation and there is an inner product $\langle \cdot \,|\, \cdot \rangle$ defined on V, it is always possible to define a new inner product on V so that each $\rho(g)$ is unitary; we will always work under this assumption. In particular, we shall always assume an orthonormal basis for V in which the matrices corresponding to $\rho(g)$ are unitary. We let $\rho(g)_{ij}$ denote the i,jth entry of the matrix for $\rho(g)$ in this fixed basis. (See, e.g., [26] for more discussion.)

We say that two representations $\rho_1 : G \to \mathrm{GL}(V)$ and $\rho_2 : G \to \mathrm{GL}(W)$ of a group G are *isomorphic* when there is a linear isomorphism of the two vector spaces $\phi : V \to W$ so that for all $g \in G$ the diagram

$$\begin{array}{ccc} V & \xrightarrow{\rho_1(g)} & V \\ {\scriptstyle\phi}\downarrow & & {\scriptstyle\phi}\downarrow \\ W & \xrightarrow{\rho_2(g)} & W \end{array}$$

commutes. That is, for all $g \in G$, $\phi\rho_1(g) = \rho_2(g)\phi$. In this case, we write $\rho_1 \cong \rho_2$. Up to isomorphism, a finite group has a finite number of irreducible representations; we let $\hat{G}$ denote this collection (of representations).

Irreducibility. We say that a subspace $W \subset V$ is an *invariant* subspace of a representation $\rho : G \to \mathrm{GL}(V)$ if $\rho(g)W \subseteq W$ for all $g \in G$. The zero subspace and the subspace V are always invariant. If no nonzero proper subspaces are invariant, the representation is said to be *irreducible*.

Decomposition and reducibility. When a representation *does* have a nonzero proper invariant subspace $V_1 \subsetneq V$, it is always possible to find a complementary subspace V_2 (so that $V = V_1 \oplus V_2$) that is also invariant. Since V_1 is invariant, for each $g \in G$, $\rho(g)$ defines a linear map $\rho_1(g)$ from V_1 to V_1 by restriction, and it is not hard to see that $\rho_1 : G \to \mathrm{GL}(V_1)$ is in fact a representation. Similarly, define $\rho_2(g)$ to be $\rho(g)$ restricted to V_2. As $V = V_1 \oplus V_2$, the linear map $\rho(g)$ is completely determined by $\rho_1(g)$ and $\rho_2(g)$, and in this case we write $\rho = \rho_1 \oplus \rho_2$. Repeating this process, any representation ρ may be written $\rho = \rho_1 \oplus \rho_2 \oplus \cdots \oplus \rho_k$, where each ρ_i is irreducible. In particular, there is a basis in which every matrix $\rho(g)$ is block diagonal, the ith block corresponding to the ith representation in the decomposition. While this decomposition is not, in general, unique, the *number* of times a given irreducible representation appears in this decomposition (up to isomorphism) depends only on the original representation ρ.

Characters. The *character* $\chi_\rho : G \to \mathbb{C}$ of a representation ρ is defined by $\chi_\rho(g) = \mathbf{tr}\,(\rho(g))$, where $\mathbf{tr}\,(\cdot)$ denotes the trace. This function is basis independent and, as it turns out, completely determines the representation ρ. Elementary properties of trace imply that characters are in fact *class* functions, depending

only on the conjugacy class of their argument. (Specifically, for every g and h we have $\chi_\rho(hgh^{-1}) = \chi_\rho(g)$.)

Orthogonality. For two complex-valued functions f_1 and f_2 on a group G, there is a natural inner product $\langle f_1 \,|\, f_2 \rangle_G$ given by $\frac{1}{|G|}\sum_g f_1(g) f_2(g)^*$. The matrix entries of the representations of a group G are orthogonal according to this inner product: let ρ and σ be two irreducible representations of G; then

$$\langle \rho(\cdot)_{ij} \,|\, \sigma(\cdot)_{kl} \rangle_G = \begin{cases} 0 & \text{if } \rho \ncong \sigma, \\ \delta_{ik}\delta_{jl} & \text{if } \rho = \sigma. \end{cases}$$

(It is assumed here that when $\rho = \sigma$, the same basis has been selected for each.) An immediate consequence is that if χ_σ is the character of a representation σ and χ_{ρ_i} is the character of an irreducible representation ρ_i, the inner product $\langle \chi_\sigma \,|\, \chi_{\rho_i} \rangle_G$ is precisely the number of times the representation ρ_i appears in the decomposition of σ. Note that since each ρ_i is unitary, we may write

$$\langle \chi_\rho \,|\, \chi_{\rho_i} \rangle_G = \frac{1}{|G|} \sum_{g \in G} \chi_\rho(g) \chi_{\rho_i}(g^{-1}).$$

Orthogonality of the second kind. Let C be a conjugacy class of G. As mentioned before, any character χ_ρ is fixed on any conjugacy class C; we denote this value by $\chi_\rho(C)$. It holds that

$$\sum_{\rho \in \hat{G}} |\chi_\rho(C)|^2 = \frac{|G|}{|C|}. \tag{2.1}$$

This is a special case of a more general principle, and we refer the interested reader to Sagan's excellent book [25, Theorem 1.10.3].

Restriction. A representation ρ of a group G is also automatically a representation of any subgroup H. We refer to this *restricted* representation on H as $\mathrm{Res}_H \rho$. Note that even representations that are irreducible over G may be reducible when restricted to H.

Induction. There is a dual notion, that of *induction*, whereby a representation of a subgroup $H < G$ may be induced to a representation of the whole group G. We delay discussion of this to section 3.1.

We let $\ker(\rho) = \{g \in G : \rho(g) = \mathbb{I}\}$ denote the kernel of a representation ρ. As a representation ρ is a homomorphism, $\ker(\rho)$ is always a normal subgroup of G. In fact, we shall see that any normal subgroup H of G can be written $\cap_{\rho \in I} \ker(\rho)$ for some set I of irreducible representations.

As mentioned above, any representation ρ of G may be decomposed into a direct sum of irreducible representations. In fact, reiterating the comments above, if $\rho_1, \ldots, \rho_k$ are the irreducible representations of G and χ_σ is the character of the representation σ, the value

$$n_i = \langle \chi_\sigma \,|\, \chi_{\rho_i} \rangle_G$$

is precisely the number of times the irreducible representation ρ_i appears in the decomposition of the representation σ into irreducible representations. Specifically, after a unitary change of basis, the matrices $\sigma(g)$ are block diagonal, consisting of n_1

copies of $\rho_1(g)$, followed by n_2 copies of $\rho_2(g)$, etc. We denote this state of affairs by $\sigma = n_1\rho_1 \oplus \cdots \oplus n_k\rho_k$.

There are two representations which shall play a central role in our discussion:

The trivial representation. The trivial representation **1** maps every group element $g \in G$ to the 1×1 identity matrix $\mathbb{I}$. Recalling the orthogonality relations above, the function $g \mapsto 1$ is orthogonal to $\rho(\cdot)_{ij}$ for any nontrivial irreducible representation ρ; this results in the identity

$$\sum_g \rho(g) = 0 \cdot \mathbb{I}, \tag{2.2}$$

which we record in anticipation of the proof of Theorem 1.2.

The regular representation. Fix a vector space V with an orthonormal basis consisting of vectors e_g, one for every element $g \in G$. The regular representation $\mathrm{reg}_G : G \to \mathrm{GL}(V)$ is defined by $\mathrm{reg}_G(g) : e_x \mapsto e_{gx}$ for any $x \in G$. V has dimension $|G|$ and, with the basis above, $\mathrm{reg}_G(g)$ is a permutation matrix for any $g \in G$.

An interesting fact about the regular representation is that it contains every irreducible representation of G. In particular, if $\rho_1, \ldots, \rho_k$ are the irreducible representations of G with dimensions $d_{\rho_1}, \ldots, d_{\rho_k}$, then

$$\mathrm{reg}_G = d_{\rho_1}\rho_1 \oplus \cdots \oplus d_{\rho_k}\rho_k,$$

so that the regular representation contains each irreducible representation ρ exactly d_ρ times. Counting dimensions yields an important relation between the dimensions d_ρ and the order of the group:

$$|G| = \sum_{\rho \in \hat{G}} d_\rho^2. \tag{2.3}$$

The main tool in quantum polynomial-time algorithms is the Fourier transform. When G is nonabelian, this takes the form described below.

DEFINITION 2.1. *Let $f : G \to \mathbb{C}$. The* Fourier transform of f at the irreducible representation ρ, *denoted $\hat{f}(\rho)$, is the $d_\rho \times d_\rho$ matrix*

$$\hat{f}(\rho) = \sqrt{\frac{d_\rho}{|G|}} \sum_{g \in G} f(g)\rho(g).$$

We refer to the collection of matrices $\langle \hat{f}(\rho) \rangle_{\rho \in \hat{G}}$ as the *Fourier transform* of f. Thus f is mapped into $|\hat{G}|$ matrices of varying dimensions. The total number of entries in these matrices is $\sum d_\rho^2 = |G|$, by (2.3) above. The Fourier transform is linear in f; with the constants used above ($\sqrt{d_\rho / |G|}$) it is in fact unitary, taking the $|G|$ complex numbers $\langle f(g) \rangle_{g \in G}$ to $|G|$ complex numbers organized into matrices.

A familiar case in computer science is when the group is cyclic of order n. Then the linear transformation (i.e., the Fourier transform) is a Vandermonde matrix over the nth roots of unity and the representations are all one-dimensional.

In the quantum setting we identify the state $\sum_{g \in G} f_g |g\rangle$ with the function $f : G \to \mathbb{C}$ defined by $f(g) = f_g$. In this notation, $\sum_{g \in G} f(g)|g\rangle$ is mapped under the Fourier transform to

$$\sum_{\rho \in \hat{G}} \sum_{1 \le i,j \le d_\rho} \hat{f}(\rho)_{i,j} |\rho, i, j\rangle.$$

We remind the reader that $\hat{f}(\rho)_{i,j}$ is a complex number and that when the first portion of this triple is measured, we observe $\rho \in \hat{G}$ with probability

$$\sum_{1 \le i,j \le d_\rho} \left| \hat{f}(\rho)_{i,j} \right|^2 = \left\| \hat{f}(\rho) \right\|_2^2,$$

where $\|A\|_2$ is the natural norm given by $\|A\|_2^2 = \mathbf{tr}\ A^* A$.

Let f be the indicator function of a left coset of H in G; i.e., for some $c \in G$,

$$f(g) = \begin{cases} \frac{1}{\sqrt{|H|}} & \text{if } g \in cH, \\ 0 & \text{otherwise.} \end{cases}$$

Our goal is to understand the Fourier transform of f, as this determines the probability of observing ρ. Our choice to measure only the representation ρ (and not the matrix indices) depends on the following key fact about the Fourier transform, also relevant to the abelian solution.

CLAIM 2.1. *The probability of observing ρ is independent of the coset.*

Proof. We have

$$\hat{f}(\rho) = \sqrt{\frac{d_\rho}{|G|}\frac{1}{|H|}} \sum_{h \in H} \rho(ch) = \sqrt{\frac{d_\rho}{|G|}\frac{1}{|H|}} \rho(c) \sum_{h \in H} \rho(h)$$

and, since $\rho(c)$ is a unitary matrix,

$$\left\| \hat{f}(\rho) \right\|_2^2 = \frac{d_\rho}{|G|}\frac{1}{|H|} \left\| \rho(c) \sum_{h \in H} \rho(h) \right\|_2^2 = \frac{d_\rho}{|G|}\frac{1}{|H|} \left\| \sum_{h \in H} \rho(h) \right\|_2^2. \qquad \square$$

Given this we may assume, without loss of generality, that our function f is $1/\sqrt{|H|}$ on the subgroup H itself, and zero elsewhere.

3. The probability of measuring ρ. The primary question is that of the probability of observing a given ρ in Experiment 1.2. We have seen that this is determined by the linear operator $\sum_{h \in H} \rho(h)$ and begin by showing that $\frac{1}{|H|} \sum_h \rho(h)$ is a projection.

LEMMA 3.1. *Let ρ be an irreducible representation of G. For every subgroup $H \le G$, $\frac{1}{|H|} \sum_{h \in H} \rho(h)$ is a projection operator.*

With the right basis, then, $\frac{1}{|H|} \sum_{h \in H} \rho(h)$ is diagonal, each diagonal entry being either one or zero. The probability of observing a particular representation ρ is then proportional to the rank of $\hat{f}(\rho)$.

Proof of Lemma 3.1. Given an irreducible representation ρ of G, we are interested in the sum of the matrices $\rho(h)$ over all $h \in H$. Since we evaluate only ρ on H, we may instead consider $\mathrm{Res}_H \rho$ without changing anything. As mentioned before, though ρ is irreducible (over G), $\mathrm{Res}_H \rho$ may *not* be irreducible over H. We may, however, decompose $\mathrm{Res}_H \rho$ into irreducible representations over H, writing $\mathrm{Res}_H \rho = \sigma_1 \oplus \cdots \oplus \sigma_t$ for a sequence σ_i of (possibly repeating) irreducible representations of H. In an appropriate basis $\sum_{h \in H} \rho(h)$ is then comprised of blocks, one corresponding to

each σ_i. In particular, the matrix $\sum_{h\in H}\rho(h)$ is

$$U\begin{bmatrix} \sum_{h\in H}\sigma_1(h) & 0 & \cdots & 0 \\ 0 & \sum_{h\in H}\sigma_2(h) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sum_{h\in H}\sigma_t(h) \end{bmatrix} U^\dagger \tag{3.1}$$

for some unitary transformation U and some irreducible representations σ_i of H (with possible repetitions). (Here † denotes conjugate transpose.) Recalling (2.2), each sum appearing on the diagonal is nonzero only when the irreducible representation is trivial, in which case it is $|H|$. □

As in the previous section, we let $f : G \to \mathbb{C}$ be the function $f(g) = \frac{1}{\sqrt{|H|}}$ for $g \in H$, and 0 otherwise. Then the probability of observing ρ in Experiment 1.2 is

$$\left\|(\hat{f}(\rho))\right\|_2^2 = \frac{d_\rho}{|G|}\frac{1}{|H|}\left\|\sum_{h\in H}\rho(h)\right\|_2^2 = \frac{d_\rho}{|G|}\frac{1}{|H|}|H|^2\left\langle \chi_\rho \,|\, \chi_{\mathbf{1}}\right\rangle_H$$
$$= \frac{|H|}{|G|}d_\rho\left\langle \chi_\rho \,|\, \chi_{\mathbf{1}}\right\rangle_H.$$

We record this result in the following theorem.

THEOREM 3.2. *For every subgroup $H \le G$, the probability of measuring ρ in Experiment* 1.2, *with hidden subgroup H, is*

$$\left\|\hat{f}(\rho)\right\|_2^2 = \frac{|H|}{|G|}d_\rho\left\langle \chi_\rho \,|\, \chi_{\mathbf{1}}\right\rangle_H.$$

Observe that this establishes the first part of Theorem 1.2, that the measurement probability equals the first expression of (1.2). Another consequence of the theorem is that the probability of observing a representation ρ depends only on the character of ρ restricted to H. As the characters are class functions, conjugate subgroups (two subgroups H_1 and H_2 are conjugate if $H_1 = gH_2g^{-1}$ for some $g \in G$) produce exactly the same distribution over ρ; this rules out using the paradigm of Experiment 1.2 with representations names alone to solve the HSP for any group containing a nonnormal subgroup.

3.1. Induced representations. We have discussed the restriction of a representation ρ of G to a subgroup H of G. There is a dual operation, *induction*, which extends to all of G a representation ρ defined on a subgroup H. We will only need to work with the representation induced from the trivial representation on H.

Let $G/H \stackrel{\text{def}}{=} \{\alpha_1, \ldots, \alpha_t\}$ be a canonical transversal for H, so that G can be written as the disjoint union $\alpha_1 H \cup \cdots \cup \alpha_t H$. Then the *induced* representation $\mathrm{Ind}_H^G\mathbf{1} : G \to \mathrm{GL}(W)$ is defined over the vector space W that has one basis vector $e_{[\alpha_i]}$ for each coset $\alpha_i H$. It is defined by linearly extending the rule

$$\mathrm{Ind}_H^G\mathbf{1}(g) : e_{[\alpha_i]} \to e_{[\alpha_j]},$$

where $\alpha_j H$ is the coset containing $g\alpha_i$. Observe that this representation is a permutation representation. As suggested by the notation, selection of a different transversal results in an isomorphic representation.

Example 3.1. $\mathrm{Ind}_{\{\mathbf{id}\}}^G\mathbf{1} \cong \mathrm{reg}_G$.

We now invoke a standard representation-theoretic result to obtain Theorem 1.2.

LEMMA 3.3 (a special case of Frobenius reciprocity; see [15, section 3.20]). *Let $H < G$ and let $\rho : G \to GL(V)$ be an irreducible representation of G. Then*

$$\langle \chi_{\mathbf{1}} \mid \chi_\rho \rangle_H = \left\langle \chi_{\mathrm{Ind}_H^G \mathbf{1}} \,\middle|\, \chi_\rho \right\rangle_G .$$

Combining this with Theorem 3.2 establishes Theorem 1.2.

Proof of Theorem 1.2. Theorem 3.2 asserts that the probability of measuring the representation ρ in Experiment 1.2 is $\frac{|H|}{|G|} d_\rho \langle \chi_{\mathbf{1}} \mid \chi_\rho \rangle_H$. By reciprocity, the number of times that the trivial representation of H appears in $\mathrm{Res}_H \rho$ is the same as the number of times that ρ appears in $\mathrm{Ind}_H^G \mathbf{1}$, that is,

$$\langle \chi_{\mathbf{1}} \mid \chi_\rho \rangle_H = \left\langle \chi_{\mathrm{Ind}_H^G \mathbf{1}} \,\middle|\, \chi_\rho \right\rangle_G .$$

Theorem 1.2 follows. □

4. A positive result: Normal subgroups and the core of H. In this section we show that $O(\log |G|)$ queries suffice to reconstruct any normal subgroup of G. In general, we show that for any subgroup H of G, the algorithm below outputs H^G, the *core of* H, which is the largest subgroup of H that is normal in G. As the product $H_1 H_2$ of two normal subgroups is again a normal subgroup of G, the core is well-defined. (In fact, the core is precisely $\bigcap_{g \in G} g H g^{-1}$.) The algorithm we study is the following.

ALGORITHM 4.1. *H is an arbitrary unknown subgroup of G; we are provided an efficiently computable function $f : G \to S$, which is constant on (left) cosets of H and takes distinct values on distinct cosets.*

1. *For $i = 1, \ldots, s = 4 \log_2 |G|$, run Experiment* 1.2 *and measure an irreducible representation, $\sigma_i \in \hat{G}$.*
2. *Let $N_i = \bigcap_{j=1}^{i} \ker \sigma_j$.*
3. *Output $N = N_s$.*

Recall that each $\ker \sigma_i$ is a normal subgroup of G, so that the resulting subgroup $N = N_s$ is normal. We will show that Algorithm 4.1 converges quickly to H^G with high probability in Theorem 4.3. We reduce the proof of this theorem to two lemmas, described in the following section. Two different sets of proofs of these lemmas are then presented in sections 4.2 and 4.3, one from the perspective of restricted representations, and one from the perspective of induced representations. The proof presented in terms of induction shows that the theorem is a consequence of the standard proof of the Mackey irreducibility criterion; for readers already acquainted with the criterion this approach may be more mnemonic than the elementary approach by restriction.

4.1. The general structure. As discussed above, Theorem 4.3 is a consequence of the following two lemmas.

LEMMA 4.1. *If the irreducible representation σ can be sampled by Experiment* 1.2 *(i.e., has nonzero probability), then $H^G \subseteq \ker(\sigma)$.*

This shows, in particular, that $H^G \trianglelefteq N_s \trianglelefteq \cdots \trianglelefteq N_1 \trianglelefteq N_0 = G$.

LEMMA 4.2. *For any subgroup $H \leq G$, if $N_i \not\subseteq H$, then* $\Pr[N_{i+1} = N_i] \leq 1/2$.

Before discussing the proofs of these lemmas, we show that together they imply Theorem 4.3. Observe that for a representation ρ of G, if $\ker \rho \subset H$, then we must have $\ker \rho \subset H^G$, as $\ker \rho$ is normal.

THEOREM 4.3. *Algorithm* 4.1 *returns H^G with probability at least* $1 - 2e^{-\log_2 |G|/8}$.

Proof of Theorem 4.3. Let $\mathcal{D}_H$ denote the probability distribution over irreducible representations induced by Experiment 1.2. We now apply a standard martingale bound (see [20]) to prove the theorem (based on Lemmas 4.2 and 4.1). Let $\sigma_1, \ldots, \sigma_k$ be independent random variables distributed according to $\mathcal{D}_H$ with $k = 4\log_2 |G|$. Our goal is to show that

$$\Pr[N_s \neq H^G] \leq 2e^{-\log_2|G|/8}.$$

For each $i \in \{1, \ldots, k\}$, let X_i be the indicator random variable taking value 1 if $N_i \subseteq H$ or $N_{i+1} \neq N_i$, and zero otherwise. The random variables $X_1, \ldots, X_k$ are not necessarily independent, but by Lemma 4.2, $\Pr[X_i = 0 \,|\, X_1, \ldots, X_{i-1}] \leq \frac{1}{2}$, and we may apply a martingale bound. As the variables take values in the set $\{0, 1\}$, the sum $\sum_i X_i$ satisfies the Lipschitz condition (with constant 1), and we can apply Azuma's inequality to conclude that $\sum_i X_i$ is unlikely to deviate far from its expected value, which is at least $\frac{k}{2}$. In particular, we have $\Pr\left[|\sum_i X_i - \frac{k}{2}| \geq \lambda\right] \leq 2e^{-\lambda^2/2k}$, so with $\lambda = \log_2(|G|)$ we have $\Pr\left[\sum_{i=0}^{k-1} X_i \leq \log_2(|G|)\right] \leq 2e^{-\log_2(|G|)/8}$.

Therefore, with probability at least $1 - 2e^{-\log_2(|G|)/8}$ we have $N_s \subseteq H$. As N_s is normal in G, it must be the case that $N_s \subseteq H^G$. From Lemma 4.1, $H^G \subseteq N_s$; hence $N_s = H^G$, and the algorithm converges to the correct subgroup. ∎

4.2. Restricted representations. We begin by proving Lemmas 4.1 and 4.2 from the perspective of restricted representations.

By Claim 2.1, we may assume that f is distributed over H itself without loss of generality. In this case, Lemma 3.1 implies that, up to a scalar multiple, each $\hat{f}(\sigma)$ is a projection. We begin by showing that when the subgroup H is *normal*, $\hat{f}(\sigma)$ is in fact a multiple of the identity, and is nonzero precisely when H is in the kernel of σ.

LEMMA 4.4. *Let $H \trianglelefteq G$, $\rho \in \hat{G}$ have dimension d_ρ, and let $f : G \to \mathbb{C}$ be the function*

$$f(g) = \begin{cases} \frac{1}{\sqrt{H}} & \text{if } g \in H, \\ 0 & \text{otherwise.} \end{cases}$$

Then $\hat{f}(\rho) = \lambda \cdot \mathbb{I}$, where

$$\lambda = \begin{cases} \sqrt{\frac{|H|}{|G|} d_\rho} & \text{if } H \subseteq \ker \rho, \\ 0 & \text{otherwise.} \end{cases}$$

Proof. The lemma follows from an application of Schur's lemma (see, e.g., [26]). If $H \subseteq \ker \rho$, then the lemma follows from the discussion in the previous section.

Suppose $H \not\subseteq \ker \rho$. We will show that $\hat{f}(\rho)$ must be the zero map. By Lemma 3.1 we can decompose V as $W_f \bigoplus W_a$, where $\frac{1}{|H|}\sum_{h \in H} \rho(h)$ pointwise fixes W_f and annihilates W_a. Our goal is to see that $W_f = \{0\}$. Observe that since $H \not\subseteq \ker \rho$, there is some $h_0 \in H$ for which $\rho(h_0)$ is not the identity operator and, considering that each $\rho(h)$ is unitary, their average cannot be the identity operator. (Specifically, consider a unit vector v for which $\rho(h_0)v \neq v$; note now that the average over h of $\rho(h)v$ can have unit length only if for all h_1, $\rho(h_0)v = \rho(h_1)v \neq v$.) Hence $W_f \neq V$.

Assume that $W_f \neq \{0\}$. Since ρ is irreducible over G and $W_f \neq V$, there is a vector $w'_f \in W_f$ and $g \in G$ such that $\rho(g)w'_f \notin W_f$; we may write $\rho(g)w'_f = w_f + w_a$,

with $w_a \in W_a$, $w_f \in W_f$, and $w_a \neq 0$. As $\frac{1}{|H|}\sum_{h\in H}\rho(h)$ pointwise fixes W_f we have

$$w_f + w_a = \rho(g)w'_f = \rho(g)\frac{1}{|H|}\sum_{h\in H}\rho(h)w'_f$$
$$\stackrel{*}{=} \frac{1}{|H|}\sum_{h\in H}\rho(h)\rho(g)w'_f = \frac{1}{|H|}\sum_{h\in H}\rho(h)(w_f + w_a) = w_f,$$

where equality $\stackrel{*}{=}$ follows since H is normal in G. This is a contradiction; hence $W_f = \{0\}$, as desired. □

We will now prove Lemma 4.1, which states that if the irreducible representation σ can be sampled by Experiment 1.2, then $H^G \subseteq \ker(\sigma)$.

Proof of Lemma 4.1. Let C be a set of coset representatives for H^G in H. We have

$$\sum_{h\in H}\rho(h) = \Big(\sum_{c\in C}\rho(c)\Big)\Big(\sum_{h\in H^G}\rho(h)\Big),$$

so by Lemma 4.4 we only observe ρ if $\sum_{h\in H^G}\rho(h)$ is a multiple of the identity (in which case $H^G \subseteq \ker\rho$). □

Before proving Lemma 4.2, which is for general subgroups, we will show how the statement can be proved for normal subgroups.

LEMMA 4.5. *If $H \trianglelefteq G$ and $N_i \neq H$, then* $\Pr[N_{i+1} = N_i] \leq 1/2$.

Proof. By Lemma 4.4, Theorem 3.2, and (2.3) we have

$$\Pr[N_i \subseteq \ker\rho_{i+1}] = \sum_{\rho\in\hat{G}N_i\subseteq\ker\rho}\Pr[\text{Observe }\rho]$$
$$= \sum_{\rho\in\hat{G}N_i\subseteq\ker\rho}\frac{|H|}{|G|}d_\rho^2$$
$$= \frac{|H|}{|G|}\sum_{\rho\in\widehat{G/N_i}}d_\rho^2 = \frac{|H|}{|G|}\frac{|G|}{|N_i|} \leq \frac{1}{2},$$

where changing the sum follows from the fact that representations of G that map N_i to the identity can be identified with representations of G/N_i. □

We proceed to prove Lemma 4.2, which states that for any $H \leq G$, if $N_i \not\subseteq H$, then $\Pr[N_{i+1} = N_i] \leq 1/2$.

Proof of Lemma 4.2. This proof is due to Umesh Vazirani. Let $N = \cap_{j=1}^{i}\ker\sigma_j$ be the intersection of the kernels up to step i. For an irreducible representation ρ, let r_ρ be the rank of $\hat{f}(\rho)$, i.e., the number of times the trivial representation of H appears in ρ. When $N \not\subseteq H$, we will show that the probability of N being contained in the kernel of the next representation we measure is at most $1/2$ by showing that

$$\sum_{\rho:N\subseteq\ker\rho}\frac{|H|}{|G|}d_\rho r_\rho \leq \frac{|N\cap H|}{|N|},$$

which is at most $1/2$ when $N \not\subseteq H$. Now, if the hidden subgroup had been HN, Theorem 3.2 would imply

$$\sum_{\rho:\rho\in\hat{G}}\frac{|HN|}{|G|}d_\rho r'_\rho = 1,$$

where r'_ρ is the number of times the trivial representation of HN appears in ρ. Note that $r'_\rho = r_\rho$ when $N \subseteq \ker \rho$, since

$$|H \cap N| \sum_{l \in HN} \rho(l) = \Big(\sum_{h \in H} \rho(h) \Big) \Big(\sum_{n \in N} \rho(n) \Big),$$

and $\rho(n)$ is the identity. Since $|HN| \cdot |H \cap N| = |H| \cdot |N|$, we have that

$$\sum_{\rho : N \subseteq \ker \rho} \frac{|H|}{|G|} d_\rho r_\rho = \sum_{\rho : N \subseteq \ker \rho} \frac{|H|}{|G|} d_\rho r'_\rho \leq \sum_{\rho \in \hat{G}} \frac{|H|}{|G|} d_\rho r'_\rho \leq \frac{|H \cap N|}{|N|},$$

as desired. ∎

4.3. Induced representations. We now reprove these two lemmas from the perspective of induced representations. We begin by computing the kernel of the representation $\mathrm{Ind}_H^G \mathbf{1}$.

LEMMA 4.6. $\ker(\mathrm{Ind}_H^G \mathbf{1}) = H^G$.

Proof. We begin with the forward inclusion. Indeed, if $x \in \ker(\mathrm{Ind}_H^G \mathbf{1})$, then $\mathrm{Ind}_H^G \mathbf{1}(x)$ is the identity mapping, i.e., for every $g \in G$, $\mathrm{Ind}_H^G \mathbf{1}(x) : [gH] \to [gH]$, or equivalently, $[xgH] = [gH]$. In particular, for $g = e$ we get $xH = H$, and therefore $x \in H$. Now, as $\ker(\mathrm{Ind}_H^G \mathbf{1})$ is normal and is contained in H we must have $\ker(\mathrm{Ind}_H^G \mathbf{1}) \subseteq H^G$.

For the reverse inclusion, suppose that $x \in H^G$. Then for any $g \in G$, there is some $x' \in H^G \subseteq H$ such that $xg = gx'$. Therefore, $\mathrm{Ind}_H^G \mathbf{1}(x)[gH] = [xgH] = [gx'H] = [gH]$, and we see that $\mathrm{Ind}_H^G \mathbf{1}(x)$ is the identity mapping. Hence, $x \in \ker(\mathrm{Ind}_H^G \mathbf{1})$. ∎

Now, by Theorem 1.2, any σ that can be sampled by Experiment 1.2 appears in $\mathrm{Ind}_H^G \mathbf{1}$, and we therefore conclude that $H^G \subseteq \ker(\mathrm{Ind}_H^G \mathbf{1}) \subseteq \ker(\sigma)$; Lemma 4.1 follows. This also gives a simple decomposition of $\mathrm{Ind}_H^G \mathbf{1}$ when H is normal.

LEMMA 4.7. *Let $N \trianglelefteq G$. Then* $\mathrm{Ind}_N^G \mathbf{1} = \bigoplus_{\rho \in \hat{G}, N \subseteq \ker(\rho)} d_\rho \rho$.

Proof. Suppose $\mathrm{Ind}_N^G \mathbf{1} = \bigoplus_{\rho \in \hat{G}} n_\rho \rho$. We have

$$n_\rho = \Big\langle \chi_{\mathrm{Ind}_N^G \mathbf{1}} \Big| \chi_\rho \Big\rangle_G \;=\; \langle \chi_{\mathbf{1}} \,|\, \chi_\rho \rangle_N \;=\; \frac{1}{|N|} \sum_{x \in N} \chi_\rho(x) \;=\; d_\rho,$$

where the second equality is by Frobenius reciprocity, and the last one is because $N \subseteq \ker \rho$. Note that $n_\rho = 0$ if $N \not\subset \ker \rho$. ∎

We now prove Lemma 4.2.

Proof of Lemma 4.2. Denote $N = N_i$. For $\rho \in \hat{G}$, let $m_\rho = \langle \chi_\rho \,|\, \chi_{\mathbf{1}} \rangle_H$. We know that

$$\Pr_{\sigma \in \mathcal{D}_H} [N_i \subseteq \ker \sigma] = \frac{|H|}{|G|} \sum_{\substack{\rho \in \hat{G} \\ N_i \subseteq \ker(\rho)}} m_\rho d_\rho,$$

$$\mathrm{Ind}_H^G \mathbf{1} = \bigoplus_{\rho \in \hat{G}} m_\rho \rho,$$

$$\mathrm{Ind}_N^G \mathbf{1} = \bigoplus_{\substack{\rho \in \hat{G} \\ N \subseteq \ker(\rho)}} d_\rho \rho,$$

where the first equation is by Theorem 3.2, the second because $m_\rho = \langle \chi_\rho \,|\, \chi_{\mathbf{1}} \rangle_H = \langle \chi_{\mathrm{Ind}_H^G \mathbf{1}} | \chi_\rho \rangle_G$ by Frobenius reciprocity (Lemma 3.3), and the last one by Lemma 4.7. We observe that

$$\left\langle \chi_{\mathrm{Ind}_H^G \mathbf{1}} \,\middle|\, \chi_{\mathrm{Ind}_N^G \mathbf{1}} \right\rangle_G = \sum_{\substack{\rho \in \hat{G} \\ N \subseteq \ker(\rho)}} d_\rho m_\rho \langle \chi_\rho \,|\, \chi_\rho \rangle_G = \sum_{\substack{\rho \in \hat{G} \\ N \subseteq \ker(\rho)}} d_\rho m_\rho$$

and thus is proportional to the probability that $N_i \subseteq \ker(\sigma_i)$. We complete the proof with an argument similar to that given in Serre [26] for the proof of Mackey's irreducibility criterion. By Frobenius reciprocity,

$$\left\langle \chi_{\mathrm{Ind}_H^G \mathbf{1}} \,\middle|\, \chi_{\mathrm{Ind}_N^G \mathbf{1}} \right\rangle_G = \left\langle \chi_{\mathbf{1}} \,\middle|\, \chi_{\mathrm{Res}_H \ \mathrm{Ind}_N^G \mathbf{1}} \right\rangle_H .$$

Decomposing the restricted induction (see [26, section 7.3]) we have

$$\left\langle \chi_{\mathrm{Ind}_H^G \mathbf{1}} \,\middle|\, \chi_{\mathrm{Ind}_N^G \mathbf{1}} \right\rangle_G = \bigoplus_{g \in H \backslash G / N} \left\langle \chi_{\mathbf{1}} \,\middle|\, \chi_{\mathrm{Ind}_{N_g}^H \mathbf{1}} \right\rangle_H ,$$

where $N_g = H \cap gNg^{-1}$ is a subgroup of H, and g runs over all representatives of the double cosets $H \setminus g/N$ of G. Using Frobenius reciprocity again we see that

$$\begin{aligned} \left\langle \chi_{\mathrm{Ind}_H^G \mathbf{1}} \,\middle|\, \chi_{\mathrm{Ind}_N^G \mathbf{1}} \right\rangle_G &= \bigoplus_{g \in H \backslash G / N} \langle \chi_{\mathbf{1}} \,|\, \chi_{\mathbf{1}} \rangle_{N_g} \\ &= |H \setminus G / N|. \end{aligned}$$

However, N is normal in G. Hence for any $g \in G$, $H \backslash g/N = HNg$. Furthermore, as H is a group and N is normal in G, HN is also a group. Hence, $|H \setminus G/N| = |G|/|HN|$. Thus,

$$\Pr_{\sigma \in \mathcal{D}_H} [N_i \subseteq \ker \sigma] = \frac{|H|}{|G|} \sum_{\substack{\rho \in \hat{G} \\ N_i \subseteq \ker(\rho)}} m_\rho d_\rho = \frac{|H|}{|G|} \frac{|G|}{|HN|} = \frac{|H|}{|HN|},$$

which is at most $1/2$ when $N \not\subseteq H$. ∎

5. A negative result: Determining triviality in S_n. In this section we show that a well-known reduction of graph isomorphism for finding a hidden subgroup over S_n cannot work using Experiment 1.2.

Graph automorphism is the problem of determining whether a graph G has a nontrivial automorphism and is easier than graph isomorphism [18]. A natural special case occurs when the graph G consists of two disjoint connected rigid graphs G_1, G_2 (i.e., $\mathrm{Aut}(G_1) = \mathrm{Aut}(G_2) = \{e\}$). In this case there are two possibilities for the automorphism group of G.

CLAIM 5.1. *Let the graph G be the disjoint union of the two connected rigid graphs G_1 and G_2 and let n denote the number of vertices of G. Then*

1. *if $G_1 \not\approx G_2$, then $\mathrm{Aut}(G) = \{e\}$, and*
2. *if $G_1 \approx G_2$, then $\mathrm{Aut}(G) = \{e, \sigma\}$, where $\sigma \in S_n$ is a permutation with $n/2$ disjoint 2-cycles.*

Proof. For the first part notice that any automorphism maps a connected component onto a connect component. In our case we have two connected components

G_1 and G_2. However, G_1 and G_2 are not isomorphic and have no nontrivial automorphisms.

For the second part, let σ reflect an automorphism between G_1 and G_2. Now, suppose there was another nontrivial automorphism τ. Then $\sigma\tau$ is also an automorphism, and $\sigma\tau$ maps the connected component of G_1 onto G_1, and G_2 onto G_2. As G_1 and G_2 have no nontrivial automorphisms it follows that $\sigma\tau = 1$, $\tau = \sigma^{-1} = \sigma$. □

Thus, if one knows how to solve the HSP for S_n, or if one knows how to distinguish between cosets of a trivial subgroup and the cosets of a nontrivial subgroup, one can give an efficient quantum algorithm for graph automorphism. In particular, one might attempt to reconstruct $H = \mathrm{Aut}(G)$ based on the result of the following experiment.

EXPERIMENT 5.1. *Let G be a graph such that either $\mathrm{Aut}(G) = \{e\}$ or $\mathrm{Aut}(G) = \{e, \sigma\}$.*

1. *Compute $\Sigma_{\pi\in S_n}|\pi, \pi(G)\rangle$ and measure the second register $\pi(G)$. The resulting state is $\Sigma_{h\in H}|ch, f(ch)\rangle$ for some coset cH of H. Furthermore, c is uniformly distributed over G.*
2. *Compute the Fourier transform of the coset state, which is*

$$\sum_{\rho\in\hat{G}} \sqrt{\frac{d_\rho}{|G|}} \sqrt{\frac{1}{|H|}} \Bigg(\sum_{h\in H} \rho(ch) \Bigg)_{i,j} |\rho, i, j\rangle.$$

3. *Measure the first register and observe a representation ρ.*

We show that even for this particular case of graph isomorphism (and graph automorphism) the experiment fails to distinguish nonisomorphic pairs of graphs from isomorphic pairs of graphs; Theorem 1.4 follows.

THEOREM 5.1. *Let G_1 and G_2 be two rigid, connected graphs with n vertices. Let $\mathcal{D}_N(\rho)$ be the probability of sampling ρ in Experiment 5.1 when $G_1 \not\approx G_2$, and let $\mathcal{D}_I(\rho)$ be the probability when $G_1 \approx G_2$. Then $|\mathcal{D}_N - \mathcal{D}_I|_1 \le 2^{-\Omega(n)}$.*

Proof. We present the proof from [10], which simplifies the proof of [14]. When $G_1 \not\approx G_2$, $H = \{e\}$, so $\mathcal{D}_N(\rho) = d_\rho^2/n!$ by Theorem 3.2. When $G_1 \approx G_2$, and G_1 and G_2 are both connected and rigid, $H = \{e, \tau\}$. By Theorem 3.2,

$$\mathcal{D}_I(\rho) \;=\; \frac{|H|}{|G|} d_\rho \left\langle \chi_{\mathbf{1}} \,|\, \chi_\rho \right\rangle_H .$$

The subgroup H has only two elements, e and τ; hence

$$\left\langle \chi_{\mathbf{1}} \,|\, \chi_\rho \right\rangle_H = \frac{1}{2}(\chi_\rho(e) + \chi_\rho(\tau)) = \frac{1}{2}(d_\rho + \chi_\rho(\tau)).$$

That is, $\mathcal{D}_I(\rho) = \frac{d_\rho}{n!}(d_\rho + \chi_\rho(\tau))$, and so

$$\begin{aligned} \sum_\rho |\mathcal{D}_I(\rho) - \mathcal{D}_N(\rho)| &= \frac{1}{n!} \sum_\rho d_\rho |\chi_\rho(\tau)| \\ &\le \frac{1}{n!} \sqrt{\sum_\rho d_\rho^2} \sqrt{\sum_\rho |\chi_\rho(\tau)|^2} = \frac{1}{\sqrt{n!}} \sqrt{\sum_\rho |\chi_\rho(\tau)|^2} \end{aligned}$$

by the Cauchy–Schwarz inequality and (2.3).

By (2.1), $\sum_{\rho\in\hat{G}} |\chi_\rho(\tau)|^2 = |G|/|\{\tau\}|$, where $\{\tau\}$ is the conjugacy class of τ. However, two permutations share the same conjugacy class if and only if they have

the same cycle decomposition. In our case τ has cycle decomposition into $n/2$ pairs. Thus

$$|\{\tau\}| = \frac{\binom{n}{n/2}(\frac{n}{2})!}{2^{n/2}},$$

where $\binom{n}{n/2}$ is the number of possibilities for choosing the first element in each of the $n/2$ pairs, $(\frac{n}{2})!$ is the number of possibilities for arranging the remaining $n/2$ elements in the pairs, and each ordering is counted exactly $2^{n/2}$ times.

Altogether,

$$\sum_{\rho} |\mathcal{D}_I(\rho) - \mathcal{D}_N(\rho)| \leq \frac{1}{\sqrt{n!}}\sqrt{\frac{n!}{|\{\tau\}|}} \;=\; \sqrt{\frac{2^{(n/2)}(n/2)!}{n!}} \;\leq\; 2^{-\Omega(n)},$$

as desired. □

6. Finding hidden subgroups in Hamiltonian groups. A group G is Hamiltonian if all subgroups are normal. In light of Theorem 3.2, a hidden subgroup of a Hamiltonian group G is determined with high probability by $O(\log |G|)$ samples of the distribution induced by Experiment 1.2. In this section we show that for Hamiltonian groups, generators for the hidden subgroup can be computed efficiently from these samples. As the Fourier transform over such groups can be efficiently computed, this gives an efficient quantum algorithm for the HSP over Hamiltonian groups.

All abelian groups are Hamiltonian; the only nonabelian Hamiltonian groups are of the form

$$G \cong \mathbb{Z}_2^k \times B \times Q,$$

where $Q = \{\pm 1, \pm i, \pm j, \pm k\}$ is the quaternion group and B is an abelian group with exponent b coprime with 2. For a detailed description of such groups, see Rotman's excellent book [23].

We begin by briefly discussing the case when G is abelian. If G is simply the cyclic group $\mathbb{Z}_n$, the representations are the functions $\rho_s : z \mapsto \exp(2\pi isz/n)$, and the reconstruction algorithm, when it succeeds, yields a collection $\{\rho_s | s \in S\}$ with the property that $H = \cap_{s\in S} \ker \rho_s$. Observe that $\rho_s(h)\rho_t(h) = \rho_{s+t \bmod n}(h)$ and that $\rho_s(h) = 1$ implies $\rho_{st \bmod n}(h) = 1$ for all $t \in \mathbb{Z}$. Hence $\cap_s \ker \rho_s = \ker \rho_d$, where d is the greatest common divisor of n and the elements in S. Then H is the cyclic subgroup of $\mathbb{Z}_n$ generated by n/d.

In general, an abelian group G is isomorphic to a direct sum $\mathbb{Z}_{n_1} \oplus \cdots \oplus \mathbb{Z}_{n_k}$, and we assume that this decomposition is known. The irreducible representations are the functions $\rho_{s_1,\ldots,s_k}(z_1, \ldots, z_k) = \prod_{j=1}^{k} \exp(2\pi i s_j z_j/n_j)$, and, as above, we begin with a collection $\{\rho_{\vec{s}} \mid \vec{s} = (s_1, \ldots, s_k) \subset S\}$ so that $H = \cap_S \ker \rho_{\vec{s}}$. Then

$$\begin{aligned} (h_1, \ldots, h_k) \in H \quad &\Leftrightarrow \quad \text{for all } \vec{s} \in S,\ \exp\left(\sum_j \frac{2\pi i s_j h_j}{n_j}\right) = 1 \\ &\Leftrightarrow \quad \text{for all } \vec{s} \in S,\ \sum_j s_j q_j h_j \equiv 0 \bmod N, \end{aligned} \tag{6.1}$$

where N is the least common multiple of the n_j, and $q_j = N/n_j$. For convenience, we treat the family of equalities appearing in line (6.1) as a system of equations over

the ring $\mathbb{Z}_N$; then a solution $\vec{h}$ of this system corresponds to the element ($h_1 \bmod n_1, \ldots, h_k \bmod n_k$) of H. Collect these equations together into a matrix R. Though $\mathbb{Z}_N$ may not be a field, it is easy to check that a matrix over $\mathbb{Z}_N$ may be diagonalized in polynomial time with the following two operations:

- for some pair $i \neq j$, swap row (column) i with row (column) j;
- for some pair $i \neq j$, add a multiple of row (column) i to row (column) j.

This results in a system $D \cdot F \cdot \vec{h} = \vec{0}$, where D is diagonal and F is invertible. Any vector $\vec{h}'$ for which $D\vec{h}' = \vec{0}$ may then be transformed into a solution $\vec{h}$ of the original equation and, moreover, if $\vec{h}'$ is selected at random in the null space of D, then the resulting $\vec{h}'$ will give rise to a random element of H. Selection of $O(\log |G|)$ random elements in this way yields a generating set for H with high probability.

Finally, consider a Hamiltonian group of the form $G = \mathbb{Z}_2^k \times B \times Q$. An irreducible representation ρ of G is a tensor product $\zeta \otimes \beta \otimes \kappa$, where $\zeta \in \widehat{\mathbb{Z}_2^k}$, $\beta \in \hat{B}$, and $\kappa \in \hat{Q}$. (As $\mathbb{Z}_2^k$ and B are abelian, in this case the tensor product may be replaced with the regular product in $\mathbb{C}$.)

We briefly review the representation theory of the quaternion group. Q has five irreducible representations: four one-dimensional and one two-dimensional. The one-dimensional representations arise as the irreducible representations of the abelian quotient $Q/\{\pm 1\} \cong \mathbb{Z}_2 \oplus \mathbb{Z}_2$. The two-dimensional representation τ realizes Q as a subgroup of $\mathbf{SU}_2$, where

$$\tau(1) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \tau(i) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix},$$
$$\tau(j) = \begin{bmatrix} i & 0 \\ 0 & -i \end{bmatrix}, \qquad \tau(k) = \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix},$$

and $\tau(-q) = -\tau(q)$ for each $q \in Q$.

As above, we assume that we have a set of samples S so that

$$H = \bigcap_{\zeta \otimes \beta \otimes \kappa \in S} \ker(\zeta \otimes \beta \otimes \kappa).$$

It is sufficient to show that for a given element $q \in Q$, one can generate a collection of random elements of $H \cap \mathbb{Z}_2^k \times B \times \{q\}$, for if these collections are large enough, then their union yields a set of generators for H with high probability.

Fixing an element $q \in Q$, consider a specific sample $\zeta \otimes \beta \otimes \kappa$. There are two cases to consider:

- If κ is one-dimensional, the condition $\zeta(z) \otimes \beta(b) \otimes \kappa(q) = 1$ may be interpreted as an equation over $\mathbb{Z}_N$, where $N = b2^{k+1}$, as in the abelian case above. (Note that $\kappa(q) = \pm 1$ contributes a constant to the equation; as $2 \mid N$, this constant can be suitably represented as $\exp(2\pi i t/N)$ for $t = N/2$.)
- If κ is two-dimensional, the condition

  $$\zeta(z) \otimes \beta(b) \otimes \kappa(q) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

  cannot be satisfied unless $q = \pm 1$. When $q = \pm 1$, this may be interpreted as a pair of equations over $\mathbb{Z}_N$, where $N = 2^{k+1}b$, each equation corresponding to a diagonal entry of the matrix. (Note that $\kappa(q)$ contributes a constant ± 1 to each equation; as $2 \mid N$, these constants can be suitably represented as $\exp(2i\pi t/N)$ for $t = N/2$.)

Now the solution proceeds as in the abelian case. For each $q \in Q$, the above procedure is used to compute $c \log |G|$ random elements of $H \cap \mathbb{Z}_2^n \times B \times \{q\}$ (unless this intersection is empty). If c is chosen appropriately, the union of these sets generates H with high probability.

Acknowledgment. The authors thank Umesh Vazirani for many helpful discussions and the simplification of several of the proofs.

REFERENCES

[1] S. Aaronson, *Quantum lower bounds for the collision problem*, in Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing, Montreal, Quebec, Canada, 2002, Association for Computing Machinery, New York, 2002, pp. 635–642.

[2] R. Beals, *Quantum computation of Fourier transforms over symmetric groups*, in Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, Association for Computing Machinery, New York, 1997, pp. 48–53.

[3] D. Boneh and R. Lipton, *Quantum cryptanalysis of hidden linear functions (extended abstract)*, in Advances in Cryptology—CRYPTO '95, D. Coppersmith, ed., Lecture Notes in Comput. Sci. 963, Springer-Verlag, Berlin, 1995, pp. 424–437.

[4] R. Cleve and J. Watrous, *Fast parallel circuits for the quantum Fourier transform*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 526–536.

[5] M. Ettinger and P. Høyer, *A Quantum Observable for the Graph Isomorphism Problem*, Los Alamos preprint, quant-ph/9901029, 1999; available online from http://lanl.arxiv.org/abs/quant-ph/9901029.

[6] M. Ettinger and P. Høyer, *Quantum State Detection via Elimination*, Los Alamos preprint, quant-ph/9905099, 1999; available online from http://lanl.arxiv.org/abs/quant-ph/9905099.

[7] M. Ettinger and P. Høyer, *On quantum algorithms for noncommutative hidden subgroups*, Adv. Appl. Math., (2000), pp. 239–251.

[8] M. Ettinger, P. Høyer, and E. Knill, *Hidden Subgroup States Are Almost Orthogonal,* Los Alamos preprint, quant-ph/9901034, 1999; available online from http://lanl.arxiv.org/abs/quant-ph/9901034.

[9] M. Goldmann and A. Russell, *The complexity of solving equations over finite groups*, Inform. Comput., 178 (2002), pp. 253–262.

[10] M. Grigni, L. Schulman, M. Vazirani, and U. Vazirani, *Quantum mechanical algorithms for the nonabelian hidden subgroup problem*, in Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, Crete, Greece, 2001, Association for Computing Machinery, New York, 2001, pp. 68–74.

[11] L. Hales and S. Hallgren, *Quantum fourier sampling simplified*, in Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, Atlanta, Georgia, 1999, Association for Computing Machinery, New York, 1999, pp. 330–338.

[12] L. Hales and S. Hallgren, *An improved quantum fourier transform algorithm and applications*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 515–525.

[13] S. Hallgren, *Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem*, in Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing, Montreal, Quebec, Canada, 2002, Association for Computing Machinery, New York, 2002, pp. 653–658.

[14] S. Hallgren, A. Russell, and A. Ta-Shma, *Normal subgroup reconstruction and quantum computation using group representations*, in Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, Portland, Oregon, 2000, Association for Computing Machinery, New York, 2000.

[15] J. Harris and W. Fulton, *Representation Theory*, Grad. Texts in Math. 129, Springer-Verlag, New York, 1991.

[16] G. Ivanyos, F. Magniez, and M. Santha, *Efficient quantum algorithms for some instances of the nonabelian hidden subgroup problem*, in Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, Heraklion, Crete Island, Greece, 2001, Association for Computing Machinery, New York, 2001, pp. 263–270.

[17] A. Kitaev, *Quantum computations: Algorithms and error correction*, Russian Math. Surveys, 52 (1997), pp. 1191–1249.
[18] J. Köbler, U. Schöning, and J. Torán, *The Graph Isomorphism Problem: Its Structural Complexity*, Progr. Theoret. Comput. Sci., Birkhäuser Boston, Boston, 1993.
[19] M. Mosca and A. Ekert, *The hidden subgroup problem and eigenvalue estimation on a quantum computer*, in Proceedings of the 1st NASA International Conference on Quantum Computing and Quantum Communications, C. Williams, ed., Lecture Notes in Comput. Sci. 1509, Springer-Verlag, Berlin, 1999, pp. 174–188.
[20] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.
[21] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, UK, 2000.
[22] M. Püschel, M. Rötteler, and T. Beth, *Fast quantum fourier transforms for a class of non-abelian groups*, in Proceedings of Applied Algebra Algebraic Algorithms, and Error-Correcting Codes (AAECC-13), Lecture Notes in Comput. Sci. 1719, Springer-Verlag, Berlin, 1999, pp. 148–159.
[23] J. Rotman, *An Introduction to the Theory of Groups*, 4th ed., Grad. Texts in Math. 149, Springer-Verlag, Berlin, 1995.
[24] M. Rötteler and T. Beth, *Polynomial-Time Solution to the Hidden Subgroup Problem for a Class of Non-Abelian Groups,* Los Alamos preprint, quant-ph/9812070, 1998; available online from http://lanl.arxiv.org/abs/quant-ph/9812070.
[25] B. E. Sagan, *The Symmetric Group*, Wadsworth and Brooks/Cole, Pacific Grove, CA, 1991.
[26] J.-P. Serre, *Linear Representations of Finite Groups*, Springer-Verlag, New York, 1977.
[27] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.
[28] D. R. Simon, *On the power of quantum computation*, SIAM J. Comput., 26 (1997), pp. 1474–1483.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 935–949

MODELS FOR RANDOM CONSTRAINT SATISFACTION PROBLEMS*

MICHAEL MOLLOY†

Abstract. We introduce a class of models for random constraint satisfaction problems. This class includes and generalizes many previously studied models. We characterize those models from our class which are asymptotically interesting in the sense that the limiting probability of satisfiability changes significantly as the number of constraints increases. We also discuss models which exhibit a sharp threshold for satisfiability in the sense that the limiting probability jumps from 0 to 1 suddenly as the number of constraints increases.

Key words. constraint satisfaction problems, random models, threshold, sharp threshold

AMS subject classification. 68R99

DOI. 10.1137/S0097539700368667

Introduction. In this paper, we introduce a new general model for generating random constraint satisfaction problems (CSPs). We will generate problems on n variables, where each variable has a domain of $d \geq 2$ permissible values, and each constraint is on k variables for some fixed $k \geq 2$. For the sake of simplicity, we assume that each variable has the same domain, $\mathcal{D} = \{\delta_1, \ldots, \delta_k\}$, although this model can be easily modified to deal with more general situations. All asymptotics will be as $n \to \infty$, and so d, k are considered to be constants. Thus, typically n will be much larger than either d or k.

This model generalizes many previously studied models, including virtually all well-studied models of CSPs, random instances of k-SAT, and CSPs which are equivalent to determining whether a random graph is k-colorable. Using straightforward methods to simulate constraints of size r by constraints of size t for any $t > r$, this model also generalizes models with mixed constraint sizes, such as the $(2 + p)$-SAT model studied in [4].

Given a k-tuple of variables, $(x_1, \ldots, x_k)$, a *restriction* on $(x_1, \ldots, x_k)$ is a k-tuple of values $R = (\delta_1, \ldots, \delta_k)$, where each $\delta_i \in \mathcal{D}$. For each k-tuple $(x_1, \ldots, x_k)$, the set of restrictions on that k-tuple is called a *constraint*. The *empty constraint* is the constraint which contains no restrictions. We say that an assignment of values to the variables of a constraint C *satisfies* C if that assignment is not one of the restrictions in C. An assignment of values to all variables in a CSP satisfies that CSP if every constraint is simultaneously satisfied. The *constraint hypergraph* of a CSP is the k-uniform hypergraph whose vertices correspond to the variables and whose hyperedges correspond to the k-tuples of variables which have *constraints*.

In studying large random CSPs one of the most natural questions is to determine the likelihood of the problem being satisfiable. For example, we might show that for some settings of the parameters in the model, a random CSP is almost surely[1] (a.s.) satisfiable, while for other settings it is a.s. nonsatisfiable. Of particular interest

*Received by the editors February 29, 2000; accepted for publication (in revised form) March 13, 2003; published electronically June 10, 2003. A preliminary version of this paper appeared in *Proceedings of the STOC* 2002 *Conference*, Montreal, Canada, 2002.
http://www.siam.org/journals/sicomp/32-4/36866.html
†Department of Computer Science, University of Toronto, Toronto, ON, M4E 3G1 Canada (molloy@cs.toronto.edu).
[1]A random CSP almost surely satisfies property P if $\lim_{n\to\infty} \mathbf{Pr}(P) = 1$.

are situations where we have a *threshold* behavior, such as a situation where, as we modify the number of constraints, a random CSP moves from being a.s. satisfiable to a.s. not satisfiable.

Some natural and important models are known to have a threshold behavior, most notably random instances of k-SAT. This fact is probably what led others to assume that the same is true of many models of random CSP. Unfortunately, and somewhat surprisingly, such assumptions often turned out to be invalid, as discussed below. Thus, a mathematically rigorous examination of the threshold behavior of models of random CSP is necessary. Such a study is the goal of the present paper.

Until recently, the most commonly used random CSP models were the following:

Model A1. Specify M, p (typically $M = cn$ for some constant c and $0 < p \leq 1$ is independent of n). First choose a random k-uniform hypergraph on n vertices with M hyperedges (where each such hypergraph is equally likely), which will be the constraint hypergraph of our problem. Next, for each hyperedge e, we choose a constraint on the k variables of e as follows: Each of the d^k possible restrictions is chosen to be present with probability p (where each choice is, of course, independent of the corresponding choices for other potential constraints).

Model A2. Specify M, m (typically $M = cn$ for some constant c and $0 < p \leq 1$ is independent of n). Choose the constraint hypergraph as in Model A1. For each hyperedge e, we choose a constraint with exactly m restrictions uniformly at random from the set of $\binom{d^k}{m}$ such constraints.

Remark 1. Alternatively, we could have chosen the constraint hypergraphs by making an independent choice for each potential hyperedge, deciding to put it in the hypergraph with probability $p = \frac{c \times k!}{n^{k-1}}$. This variation produces models which are equivalent to Models A1 and A2 in the sense that the models will be a.s. satisfiable and a.s. unsatisfiable for the same values of c. A similar remark applies to the other two models described below.

In [5], we observed that Model A1 is asymptotically uninteresting in the sense that as long as M grows with n, the random CSP will be a.s. unsatisfiable. The problem is quite simple: a.s. there will be at least one constraint which is overconstraining in that there is no assignment to its variables which satisfies even that constraint, i.e., a constraint which includes every possible restriction amongst its k variables. Model A2 avoided this particular problem but had other equally damaging problems as long as M was of order $\Theta(n)$ (as is usually the case) for all but small values of m: $m < d^{k-1}$ (Gent et al. [12] showed that these small values of m are indeed not flawed). [12] provides a survey of models used in previous research and found that roughly 3/4 of papers used models which were uninteresting in the sense described above. Furthermore, those models which were interesting (mostly Model A2 with $m < d^{k-1}$) seem to have been "accidentally" interesting in the sense that the authors were unaware that the small value they chose for m was crucial.

So we proposed an alternative model [5]:

Model B. Specify M (typically $M = cn$ for some constant c). Choose a random CSP containing M restrictions, where each such CSP is equally likely to be chosen.

We showed that this model was asymptotically interesting in the sense that for small values of c a random CSP is a.s. satisfiable, while for large values of c it is a.s. unsatisfiable.

Nevertheless, this model is rather unsatisfactory in the sense that the CSPs which it generates do not resemble CSPs that researchers tend to find interesting. In particular, almost every constraint contains only one restriction, while in typical CSPs

most constraints tend to have several restrictions.

Here, we propose a model which is both asymptotically interesting and which overcomes the main disadvantage of Model B in that it can generate CSPs whose constraints have several restrictions. We aim to keep the model as simple as possible, while at the same time meeting these two (somewhat conflicting) goals. In particular, our model is symmetric in the variables. It is not necessarily symmetric among the values of $\mathcal{D}$, although the parameters of the model can be set so that it is.

Gent et al. [12] proposed another model which is both asymptotically interesting and generates reasonably complex constraints. Their model turns out to be a special case of the model introduced here. In the terminology introduced subsequently in this paper, they proved that their model is *very well-behaved* en route to showing that it exhibits a *transition*.

Our New Model is similar in flavor to Models A1 and A2, but we must take some care in the way that we choose the constraints in order to avoid overconstrained constraints and similar problems. In particular, we cannot just independently choose whether or not to put each potential restriction in the constraint—we must choose the entire constraint en masse.

So, we consider a canonical set of variables, $X_1, \ldots, X_k$, and the set of $2^{d^k} - 1$ potential nonempty constraints on $X_1, \ldots, X_k$. We specify a probability distribution $\mathcal{P}$ which selects a single random constraint, where $\mathcal{P}$ does not depend on n.

New Model. Specify $M, \mathcal{P}$ (typically $M = cn$ for some constant c). First choose a random constraint hypergraph with M hyperedges in the usual manner. Next, for each hyperedge e, we choose a constraint on the k variables of e as follows: we take a random permutation from the k variables onto $\{X_1, \ldots, X_k\}$ and then we select a random constraint according to $\mathcal{P}$, mapping it onto a constraint on our k variables in the obvious manner.

Remark 2. Again, we could have chosen the constraint hypergraph by making an independent choice for each potential hyperedge, deciding to put it in the hypergraph with probability $p = \frac{c \times k!}{n^{k-1}}$. This variation produces a model which is, for our purposes, equivalent to the model described above.

We use $CSP_{n,M}(\mathcal{P})$ to denote a random CSP drawn from our New Model with parameters $n, M, \mathcal{P}$. We occasionally omit the subscript n, M, depending on the context. We say that $CSP(\mathcal{P})$ exhibits a *partial transition* if there exist constants $c_1, c_2 > 0$ such that

(i) if $M < c_1 n$, then $CSP_{n,M}(\mathcal{P})$ is not a.s. unsatisfiable;
(ii) if $M > c_2 n$, then $CSP_{n,M}(\mathcal{P})$ is a.s. unsatisfiable.

We say that $CSP(\mathcal{P})$ exhibits a *transition* if there exist $c_1, c_2 > 0$ such that

(i) if $M < c_1 n$, then $CSP_{n,M}(\mathcal{P})$ is a.s. satisfiable;
(ii) if $M > c_2 n$, then $CSP_{n,M}(\mathcal{P})$ is a.s. unsatisfiable.

Note that if $CSP(\mathcal{P})$ exhibits a transition, then it exhibits a partial transition. Note also that having a *transition* is a weaker property than having a "sharp threshold," a notion that will be discussed in the final section of this paper.

Remark 3. It turns out that there is no choice of $CSP(\mathcal{P})$ and c_2 such that for all $M > c_2 n$, $CSP_{n,M}(\mathcal{P})$ is neither a.s. satisfiable nor a.s. unsatisfiable. Thus there is no need for the obvious 3rd and 4th notions of "transition."

In the next two sections, we will characterize those models $CSP(\mathcal{P})$ which have transitions and partial transitions. In the final section, we will discuss choices for $\mathcal{P}$ for which $CSP(\mathcal{P})$ has a very sharp threshold of satisfiability.

We set $\mathcal{C} = \text{supp}(\mathcal{P})$, i.e., the set of constraints, C, for which $\mathcal{P}(C) > 0$. We

will see that, perhaps surprisingly, for each $\mathcal{C}$, $CSP(\mathcal{P})$ exhibits a transition (partial transition) for at least one choice of $\mathcal{P}$ with $\mathcal{C} = \text{supp}(\mathcal{P})$ iff $CSP(\mathcal{P})$ exhibits a transition (partial transition) for *every* such choice of $\mathcal{P}$. Thus, it makes sense to say that $\mathcal{C}$ exhibits a *transition* or a *partial transition*.

A constraint C is *unsatisfiable* if it has no satisfying assignments, i.e., if it contains every possible restriction. A constraint C *forbids* $X_i = \delta$ if it has no satisfying assignments for which $X_i = \delta$. $X_i = \delta$ *implies* $X_j = \gamma$ *in* C if there are no satisfying assignments for C in which $X_i = \delta$ and $X_j \neq \gamma$. A constraint C *implies* a property P if every satisfying truth assignment for C has P.

An *automorphism* of $\mathcal{C}$ is a bijection $\phi : \mathcal{D} \to \mathcal{D}$ such that $C = \{(\delta_1^1, \ldots, \delta_k^1), \ldots, (\delta_1^t, \ldots, \delta_k^t)\} \in \mathcal{C}$ iff $\phi(C) = \{(\phi(\delta_1^1), \ldots, \phi(\delta_k^1)), \ldots, (\phi(\delta_1^t), \ldots, \phi(\delta_k^t))\} \in \mathcal{C}$. We say that $\mathcal{C}$ is *symmetric with respect to* $\mathcal{D}$ if it is symmetric in the following sense: For any two values $\delta, \gamma \in \mathcal{D}$, there is an automorphism ϕ of $\mathcal{C}$ such that $\phi(\delta) = \gamma$.

Note that we do not require the symmetry to extend to $\mathcal{P}$; i.e., it is possible that $\mathcal{P}(C) \neq \mathcal{P}(\phi(C))$. Furthermore, $\mathcal{C}$ does not need to be symmetric in $X_1, \ldots, X_k$. For example, it could be that every constraint $C \in \mathcal{C}$ implies that $X_1 = X_2 \neq X_3$, while at the same time $\mathcal{C}$ is symmetric with respect to $\mathcal{D}$. However, the fact that our model takes a random mapping of the variables from an edge of the constraint hypergraph onto $(X_1, \ldots, X_k)$ implies that the model is symmetric with respect to the variables.

Our New Model generalizes several previously studied models of CSP and other problems which reduce to being random instances of CSP. For example, Model A1 is simply this model where $\mathcal{P}(C) = p^{|C|}(1-p)^{d^k - |C|}$. Model A2 is the case where $\mathcal{P}(C) = \binom{d^k}{m}^{-1}$ if $|C| = m$, and $\mathcal{P}(C) = 0$ otherwise. Model B is virtually equivalent to the case where $\mathcal{P}(C) = \frac{1}{d^k}$ if $|C| = 1$, and $\mathcal{P}(C) = 0$ otherwise. The well-studied problem of whether a random instance of k-SAT is satisfiable is equivalent to Model B where $d = 2$. The problem of deciding whether a random graph is c-colorable is equivalent to the case where $k = 2, d = c$ and $\mathcal{P}(C) = 1$ where $C = \{(\delta_1, \delta_1), \ldots, (\delta_c, \delta_c)\}$. In each of these examples, $\mathcal{C}$ is symmetric. It is already known that the first of these examples does not exhibit even a partial transition, that the second one exhibits a partial transition only if $m < d^{k-1}$, the third exhibits a transition, and the other two exhibit transitions for $k, c > 2$ and partial transitions for $k, c = 2$.

One final definition: a *hypercycle* of a k-uniform hypergraph is a set of hyperedges $e_1, \ldots, e_m$, such that each e_i contains a vertex which it shares with e_{i-1}, a different vertex which it shares with e_{i+1} (addition and subtraction are mod m), and $k - 2$ vertices which do not lie in any $e_j, j \neq i$. The *length* of the hypercycle is m, and we sometimes say that it is an m-hypercycle. If there are no other hyperedges in the hypergraph, then we say that the hypergraph itself is a hypercycle.

It is important to note that by the nature of the asymptotics involved, this study is only relevant as the number of variables grows very high. So these asymptotic results might be completely out-of-sync with experimental results which, due to technological limitations, are usually performed on a small number of variables, often less than 50. In fact, it is precisely this difference between the behavior of the model on small instances and its asymptotic behavior which led to earlier false assumptions about the asymptotic behavior of Models A1 and A2.

I have recently learned that Creignou and Daudé [8] have independently considered the special case of this New Model where $d = 2$ and $\mathcal{P}$ is the uniform distribution over $\text{supp}(\mathcal{P})$. They obtain, for those choices of $\mathcal{P}$, results which are similar to those in sections 1 and 2 below.

1. The symmetric case. In this section, we focus on the case where $\mathcal{C}$ is symmetric with respect to $\mathcal{D}$. We say that such a $\mathcal{C}$ is *well-behaved* if it satisfies the following two properties:

1. There is no $C \in \mathcal{C}$, $\delta \in \mathcal{D}$, and canonical variable X_i such that C forbids $X_i = \delta$.
2. For every $\delta \in \mathcal{D}$, there is at least one $C \in \mathcal{C}$ such that $X_1 = \delta, X_2 = \delta, \ldots, X_k = \delta$ does not satisfy C.

$\mathcal{C}$ is *very well-behaved* if, in addition to properties 1 and 2, it also satisfies the following:

3. If a CSP is formed using constraints from $\mathcal{C}$, and its constraint hypergraph is a cycle, then it must be satisfiable.

It is worth noting that if property 1 holds, then there is no unsatisfiable constraint $C \in \mathcal{C}$, and so we avoid the main problem with Model A1. However, as we will see, for $CSP(\mathcal{P})$ to not be a.s. unsatisfiable, it does not suffice to merely have no unsatisfiable constraint $C \in \mathcal{C}$—we need the full power of property 1 in order to avoid a more subtle problem.

THEOREM 1. *If $\mathcal{C}$ is well-behaved, then for every distribution $\mathcal{P}$ with $\mathcal{C} = \text{supp}(\mathcal{P})$, $CSP(\mathcal{P})$ exhibits a partial transition. If $\mathcal{C}$ is not well-behaved, then for every distribution $\mathcal{P}$ with $\mathcal{C} = \text{supp}(\mathcal{P})$, $CSP(\mathcal{P})$ does not exhibit a partial transition.*

Note that Theorem 1 immediately implies that Model A2 exhibits a partial transition iff $1 \leq m < d^{k-1}$, as first shown by Gent et al. [12].

THEOREM 2. *If $\mathcal{C}$ is very well-behaved, then for every distribution $\mathcal{P}$ with $\mathcal{C} = \text{supp}(\mathcal{P})$, $CSP(\mathcal{P})$ exhibits a transition. If $\mathcal{C}$ is not very well-behaved, then for every distribution $\mathcal{P}$ with $\mathcal{C} = \text{supp}(\mathcal{P})$, $CSP(\mathcal{P})$ does not exhibit a transition.*

Proof of Theorem 1. First we prove that our two properties are necessary for $CSP(\mathcal{P})$ to have a partial transition.

Suppose that property 2 does not hold. Then for at least one value δ, setting each $X_i = \delta$ will satisfy any $C \in \mathcal{C}$, and so setting every variable in our CSP equal to δ will satisfy our formula with probability 1, regardless of M (in fact, since $\mathcal{C}$ is symmetric in $\mathcal{D}$, this is true for every $\delta \in \mathcal{D}$). Thus, there is no value of M for which $CSP_{n,M}(\mathcal{P})$ is a.s. unsatisfiable.

Suppose that property 1 does not hold, i.e., that there is some value $\delta \in \mathcal{D}$, canonical variable X_i, and $C \in \mathcal{C}$ such that C forbids $X_i = \delta$. Since $\mathcal{C}$ is symmetric in $\mathcal{D}$, we have that for *every* $\delta \in \mathcal{D}$ there is some $C(\delta) \in \mathcal{C}$ such that C forbids $X_i = \delta$.

We say that a variable x is *impossible* if x lies in at least d hyperedges of our constraint hypergraph, $e_1^j, \ldots, e_d^j$, and each e_ℓ^j has the constraint $C(\delta_\ell)$ where x is mapped onto X_i.

It is not hard to see that if x is impossible, then our formula is not satisfiable, since x is forbidden from being assigned any of $\delta_1, \ldots, \delta_d$.

Suppose that $M = cn$ for any value of c.

Claim. A.s. $CSP_{n,M}(\mathcal{P})$ has an impossible variable.

Proof. First choose the constraint hypergraph. It is well known[2] (and an easy application of Chebyshev's Inequality) that a.s. there will be at least ζn vertices with degree exactly d for a particular constant $\zeta = \zeta(c, k, d) > 0$. Let E_1 be the event that we do have this many such vertices. If E_1 holds, then a simple greedy algorithm will produce an independent set I of size at least $\frac{\zeta}{(k-1)d+1} n$ containing only vertices of

[2]See [7] for a proof of the case $k = 2$. The proof for higher values of k is similar.

degree d: Simply add vertices of degree d to I one at a time. Each time we add a vertex to I, we forbid any of its $d(k-1)$ neighbors from being added to I. We will always be able to continue until $|I| > \frac{\zeta}{(k-1)d+1}n$.

Next we choose the constraints for each hyperedge. Each vertex $v \in I$ will become impossible with some positive probability $p_1 > 0$. Furthermore, each such event is independent of the corresponding events for the other vertices in I, since I is an independent set. Therefore, the probability that there is no impossible variable is at most $\mathbf{Pr}(\overline{E_1}) + (1-p)^{\frac{\zeta}{(k-1)d+1}n} = o(1)$.

Thus, we have shown that if $\mathcal{C}$ is not well-behaved, then either $CSP_{n,M}(\mathcal{P})$ is always a.s. unsatisfiable or it is always a.s. satisfiable, regardless of $\mathcal{P}$, and so $\mathcal{C}$ does not exhibit a partial transition.

Now we show that if $\mathcal{C}$ is well-behaved, then it exhibits a partial transition for any choice of $\mathcal{P}$. First, we prove that property 1 implies that we can take c_1 to be any constant less than $\frac{1}{k(k-1)}$; i.e., for every $c < \frac{1}{k(k-1)}$ and $M < cn$, $CSP_{n,M}(\mathcal{P})$ is a.s. satisfiable.

It is well known[3] that for $c < \frac{1}{k(k-1)}$, the probability that our constraint hypergraph is acyclic tends to a constant $\rho(c) > 0$. It is not hard to show that if our constraint hypergraph is acyclic, then property 1 implies that our CSP is satisfiable. Thus the probability of satisfiability is at least ρ, and so we are not a.s. unsatisfiable.

For each component of the constraint hypergraph, we pick a variable to be the root and set it arbitrarily. We then process each hyperedge in sequence, i.e., assign values to the variables of that hyperedge, such that every time we process a hyperedge, exactly one of its variables has already been set. Since no $C \in \mathcal{C}$ forbids any values from any variables, we will be able to find an assignment for the other variables on that hyperedge without violating the corresponding constraint.

Finally, we complete our proof by showing that property 2 implies the existence of c_2, using what is by now a standard first moment argument.

Let A be the number of satisfying assignments of our random CSP. We will prove that there exists c_2 such that if $p = c/n$ where $c > c_2$, then $\mathbf{Exp}(A) = o(1)$. This implies that a.s. $A = 0$, i.e., our CSP is unsatisfiable.

There are d^n truth assignments. Consider any one particular assignment $\mathcal{A}$. There is a value δ such that some set S of at least $\frac{n}{d}$ variables are assigned δ in $\mathcal{A}$. By property 2, there is some constraint $C^* \in \mathcal{C}$ such that setting each $X_i = \delta$ violates C^*. The probability that $\mathcal{A}$ is a satisfying assignment is at most the probability that none of the k-tuples of S is constrained by C^*. The probability that a randomly chosen constraint is C^* applied to a k-tuple of S is

$$\mathcal{P}(C^*) \times \binom{n/d}{k} / \binom{n}{k} \approx \frac{\mathcal{P}(C^*)}{d^k}.$$

From this fact, standard easy arguments yield that

$$\begin{aligned}\mathbf{Exp}(A) &\leq d^n \left(1 - \frac{\mathcal{P}(C^*)}{d^k}\right)^{cn} \\ &< \mathrm{e}^{n \times \left(\ln d - \frac{c\mathcal{P}(C^*)}{d^k}\right)} \\ &= o(1)\end{aligned}$$

[3]See [7] for a proof of the case $k = 2$. See also the branching argument inspired by Karp [13] in [6]. The proof for higher values of k is similar.

for $c > c_2 = \frac{d^k \ln d}{\mathcal{P}(C^*)}$. ∎

Remark 4. In fact, note that we have shown that our CSP is always a.s. unsatisfiable iff property 1 does not hold, and it is always a.s. satisfiable iff property 2 does not hold.

Proof of Theorem 2. First we show why property 3 is necessary for our model to have a transition. Suppose that it doesn't hold, i.e., that there is some unsatisfiable CSP formula C consisting of constraints from $\mathcal{C}$ and whose constraint hypergraph forms an ℓ-cycle.

For any value of $c > 0$, it is well known[4] that the probability that a random hypergraph with cn edges has an ℓ-cycle tends to a constant $\zeta_1 = \zeta_1(c, \ell) > 0$. If our constraint hypergraph has an ℓ-cycle, then the probability of that ℓ-cycle being constrained by C is equal to some constant $\zeta_2 > 0$. Therefore, the probability of being unsatisfiable tends to at least $\zeta_1\zeta_2 > 0$, and so we are not a.s. satisfiable for any value of c.

On the other hand, if property 3 does hold, then for $c < \frac{1}{k(k-1)}$ our CSP is a.s. satisfiable. It is well known that a.s. the constraint hypergraph will have no component with more than 1 cycle. If this happens, then the variables of each tree component can be set as in the proof of Theorem 1. The variables of each unicyclic component can be set as follows: first set the variables of the cycle in a satisfactory manner (which will always be possible by property 3), and then set the remaining variables in the same manner as in a tree component.

The remainder of the theorem follows as in the proof of Theorem 1. ∎

In our proof, we showed that if $\mathcal{C} = \text{supp}(\mathcal{P})$ exhibits a transition, then $CSP(\mathcal{P})$ is a.s. satisfiable when $M = cn$ where c is below $\frac{1}{k(k-1)}$, the threshold for the constraint hypergraph to have a giant component. Not surprisingly, for many choices of $\mathcal{P}$, the same is still true for higher values of M. For example, see the following theorem.

THEOREM 3. *Suppose that $\mathcal{C}$ is such that for every minimally unsatisfiable formula F whose constraints are drawn from C, the ratio of constraints to variables of F is at least $\frac{1}{k-1}(1+\epsilon)$ for some constant $\epsilon > 0$. Then there is a constant $\delta > 0$ such that for any $\mathcal{P}$ with $\text{supp}(\mathcal{P}) = \mathcal{C}$, $CSP(\mathcal{P})$ is a.s. satisfiable for $M \leq \frac{1}{k(k-1)}(1+\delta)n$.*

Note that for the hypothesis of Theorem 3 to hold, $\mathcal{C}$ must be very well-behaved since a cycle has a hyperedge/vertex ratio of exactly $\frac{1}{k-1}$, and a tree has an even smaller ratio.

Proof. This is a standard type of argument in random graph theory, but we include it here because it is not well known.

We can assume that ϵ is arbitrarily small, say $\epsilon \leq \frac{1}{2}$. Suppose that $M = cn$ where $c = \frac{1}{k(k-1)}(1+\delta)$, and $\delta < \epsilon$ is a small constant to be specified later. We will show that the expected number of subgraphs of our random constraint hypergraph which have the required hyperedge/vertex ratio is $o(1)$. Thus, a.s. none exist and so the CSP must be satisfiable since any unsatisfiable formula contains a minimally unsatisfiable subformula.

We can assume that such a subgraph has no tree or unicyclic components, since deleting such a component will only increase the hyperedge/vertex ratio. Thus, the subgraph must lie within the giant component. The size of the giant component is well studied[5] and is a.s. at most αn where α tends to zero with δ. Thus, we can

[4]See [7] for a proof of the case $k = 2$. The proof for higher values of k is similar.

[5]See, for example, [7] for the case $k = 2$. The case $k > 2$ is amenable to similar analysis or to the simpler branching process analysis found in [6] and inspired by [13].

restrict our attention to subgraphs of size at most αn.

For any $0 < a \leq \alpha n$, we consider the expected number of subgraphs on a vertices with $b = \frac{1}{k-1}(1+\epsilon)a$ edges. There are $\binom{a}{2}$ choices for the vertices and $\binom{M}{b}$ choices for the edges. Given such choices, the probability that all the b edges lie within the a vertices is at most $\left(\frac{a}{n}\right)^{kb}$. Therefore, the expected number of such subgraphs is at most

$$\begin{aligned}\binom{n}{a}\binom{M}{b}\left(\frac{a}{n}\right)^{kb} &\leq \left(\frac{\mathrm{e}n}{a}\right)^a \left(\frac{\mathrm{e}M}{b}\right)^b \left(\frac{a}{n}\right)^{kb} \\ &= \left(\frac{\mathrm{e}^{1+(1+\epsilon)/(k-1)}c^{(1+\epsilon)/(k-1)}(k-1)}{1+\epsilon}\left(\frac{a}{n}\right)^{\epsilon}\right)^a \\ &= \left(K\left(\frac{a}{n}\right)^{\epsilon}\right)^a,\end{aligned}$$

where $K = K(\epsilon)$ is a constant. For $a \leq n^{\epsilon/4}$, this expression is $(\Theta(n^{-\epsilon(1-\epsilon/4)}))^a < n^{-\epsilon/2}$. And for $a < \alpha n$, it is less than $(\frac{1}{2})^a$, where α is sufficiently small in terms of ϵ. Thus the expected number of subgraphs is less than

$$\sum_{a=1}^{n^{\epsilon/4}} n^{-\epsilon/2} + \sum_{a \geq n^{\epsilon/4}} \left(\frac{1}{2}\right)^a = o(1)$$

for $\delta = \delta(\alpha)$ sufficiently small. □

Theorem 3 applies to several common models. For example, a minimally non-r-colorable graph has edge/vertex ratio at least $r/2$ since such a graph has minimum degree at least r. Also, a minimally unsatisfiable instance of k-SAT has clause/variable ratio at least $2/k$ since it cannot contain any pure literals. In a future publication [14], we will characterize the distributions $\mathcal{P}$ for which $CSP(\mathcal{P})$ is a.s. unsatisfiable with $M = cn$ for all $c > \frac{1}{k(k-1)}$.

2. The asymmetric case. Here we extend the definitions and theorems of the previous section to the case where $\mathcal{C}$ is not necessarily symmetric in $\mathcal{D}$.

We say that a value δ is 0-*bad* if there is some canonical variable X_i and constraint $C \in \mathcal{C}$ such that C forbids $X_i = \delta$. We say that δ is j-*bad* if there is some canonical variable X_i and constraint $C \in \mathcal{C}$ such that C implies that if $X_i = \delta$, then at least one other canonical variable must be assigned a j'-bad value for some $j' < j$. A value is *bad* if it is j-bad for some j. A value is *good* if it is not bad.

If $\mathcal{C}$ is symmetric, then, of course, if one value is bad, then all values are bad, and so the existence of a bad value would violate property 1 from the previous section. However, to allow for the case that $\mathcal{C}$ is asymmetric, we must modify property 1. We say that $\mathcal{C}$ is *well-behaved* if it satisfies the following two properties:

1′. There is at least one good value in $\mathcal{D}$.

2. For every $\delta \in \mathcal{D}$, there is at least one $C \in \mathcal{C}$ such that $X_1 = \delta, X_2 = \delta, \ldots, X_k = \delta$ does not satisfy C.

Remark 5. It is an easy exercise to show that properties 1′ and 2 imply that there are at least 2 good values in $\mathcal{D}$.

We also must modify property 3 slightly. We say that $\mathcal{D}$ is *very well-behaved* if, in addition to properties 1′ and 2, it also satisfies the following:

3′. If a CSP is formed using constraints from $\mathcal{C}$, and its constraint hypergraph is a cycle, then it must have a satisfying assignment where no variable is assigned a bad value.

We have the analogous results from the previous section.

THEOREM 4. *If $\mathcal{C}$ is well-behaved, then for every distribution $\mathcal{P}$ with $\mathcal{C} = \text{supp}(\mathcal{P})$, $CSP(\mathcal{P})$ exhibits a partial transition. If $\mathcal{C}$ is not well-behaved, then for every distribution $\mathcal{P}$ with $\mathcal{C} = \text{supp}(\mathcal{P})$, $CSP(\mathcal{P})$ does not exhibit a partial transition.*

THEOREM 5. *If $\mathcal{C}$ is very well-behaved, then for every distribution $\mathcal{P}$ with $\mathcal{C} = \text{supp}(\mathcal{P})$, $CSP(\mathcal{P})$ exhibits a transition. If $\mathcal{C}$ is not very well-behaved, then for every distribution $\mathcal{P}$ with $\mathcal{C} = \text{supp}(\mathcal{P})$, $CSP(\mathcal{P})$ does not exhibit a transition.*

The proofs are along the same lines as those in the previous section. We present only the portions of the proofs which are different.

Proof of Theorem 4. First we show that if condition $1'$ does not hold, then for every distribution $\mathcal{P}$ and value $c > 0$, $CSP(\mathcal{P})$ is a.s. unsatisfiable.

For each $j \geq 0$ we let b_j denote the number of j-bad values. Thus $\sum_{j \geq 0} b_j = d$. For each 0-bad value δ, we let $C(\delta)$ be the constraint and $X_{i(\delta)}$ be the canonical variable such that $C(\delta)$ forbids $X_{i(\delta)} = \delta$. Similarly, if $j \geq 1$, then for each j-bad value δ, we let $C(\delta)$ be the constraint and $X_{i(\delta)}$ be the canonical variable such that if $X_{i(\delta)} = \delta$, then at least one other canonical variable must take a j'-bad value for some $j' < j$.

We define a *j-bad tree* rooted at a variable v as follows:

- A 0-bad tree rooted at v is a set of b_0 hyperedges intersecting at v such that for each 0-bad value δ, one of the hyperedges receives the constraint $C(\delta)$ with $X_{i(\delta)}$ mapped onto v.
- For $j \geq 1$, a j-bad tree rooted at v consists of a j'-bad tree $T_{j'}$ rooted at v for each $0 \leq j' < j$, along with b_j hyperedges $e_1, \ldots, e_{b_j}$ intersecting at v such that
  - (i) for each j-bad value δ, a hyperedge e_i receives the constraint $C(\delta)$ with $X_{i(\delta)}$ mapped onto v;
  - (ii) for every vertex $u \neq v$ in each e_i, u is the root of a $(j-1)$-bad tree T_u;
  - (iii) all of the bad trees $\{T_0, \ldots, T_{j-1}\} \cup \{T_u : u \text{ is in some } e_i\}$ are disjoint except that $T_0, \ldots, T_{j-1}$ all contain v.

It is not hard to argue inductively that if v roots a j-bad tree, then v cannot receive a j'-bad value for any $j' \leq j$: If $j = 0$, then for each 0-bad value δ, v is forbidden δ by $C(\delta)$. For $j \geq 1$, since for each $j' < j$, v roots a j'-bad tree, v cannot receive a j'-bad value. If v receives some j-bad value, δ, then since v lies in a $C(\delta)$-constraint, at least one of the other variables in that constraint must receive a j'-bad value for some $j < j$. But each of those variables roots a $(j-1)$-bad tree and hence roots a j'-bad tree. Thus it cannot take any j'-bad value.

A *bad tree* is a j-bad tree where j is the maximum value such that $b_j > 0$. If $1'$ does not hold, then every value is a j'-bad value for some $j' < j$, and so the root of a bad tree cannot receive any of them. Therefore, if a CSP contains a bad tree, then it is unsatisfiable.

Now, we show that for any distribution $\mathcal{P}$ and $c > 0$, a.s. $CSP(\mathcal{P})$ contains a bad tree. The following fact is well known in random graph theory.

Fact. For any fixed hypertree T independent of n, there exists a constant $z > 0$ such that a.s. our random constraint hypergraph will contain at least ζn copies of T.

The basic outline for the proof of this fact is as follows: (1) The expected number of copies of T is $\zeta' n$ by a simple but tedious calculation. (2) Because n grows much larger than the (constant) size of T, individual potential appearances of T occur nearly independently. An application of Chebyshev's Inequality formalizes this and shows

that the number of copies of T is highly concentrated. In particular, it is a.s. at least ζn for any $\zeta < \zeta'$. See [7] for details of the case $k = 2$. The analysis for higher k is the same.

Naturally, we will apply this fact where T is equal to the underlying hypertree of a minimum-sized bad tree. Let E_1 be the event that our hypergraph contains at least ζn copies of T. If E_1 holds, then using a simple greedy procedure as in the proof of Theorem 1, we can easily find a set I of $\frac{\zeta}{|T|}n$ disjoint copies of T (where $|T|$ denotes the number of vertices in T). Upon choosing the constraints, a copy becomes bad with probability p_1 for some $p_1 > 0$ which is not a function of n. Therefore, the probability that our CSP contains no bad trees is at most

$$\mathbf{Pr}(E_1) + (1 - p_1)^{\frac{\zeta}{|T|}n} = o(1).$$

To prove that if $1'$ holds, then for any $\mathcal{P}$ and $c < \frac{1}{k(k-1)}$, $CSP(\mathcal{P})$ is not a.s. unsatisfiable we just have to show that any CSP with constraints from $\mathcal{C}$ and whose constraint hypergraph is a tree is satisfiable.

For such a CSP, choose any variable v to be the root. Assign v a good value. Since v is good, it is possible to assign good values to every variable which shares a constraint with v without violating any of those constraints. Now continue through the CSP as in the proof of Theorem 1, each time assigning only good values, thus satisfying the CSP.

The remainder of the proof, namely, the relevance of property 2, is identical to the proof of Theorem 1. □

Proof of Theorem 5. This follows almost exactly as the proof of Theorem 2.

If property $3'$ does not hold, then with probability tending to some positive constant $CSP(\mathcal{P})$ will contain a subproblem whose constraint hypergraph is a unicyclic component and (i) the cycle of that component is constrained such that at least one variable must receive a bad value, and (ii) each variable on that cycle roots a bad tree. Clearly, such a subproblem is unsatisfiable.

If property $3'$ holds, then it is enough to show that any CSP with constraints from $\mathcal{C}$ and whose constraint hypergraph is unicyclic is satisfiable. To satisfy such a CSP, first assign good values to each variable on the cycle without violating any of the cycle constraints, and then continue to assign good values to each of the other variables as in the proof of Theorem 4. □

Theorem 3 also applies to the asymmetric case. Again, it would be interesting to characterize the distributions $\mathcal{P}$ for which $CSP(\mathcal{P})$ is a.s. unsatisfiable with $M = cn$ for all $c > \frac{1}{k(k-1)}$.

3. Sharp thresholds. If a model $CSP(\mathcal{P})$ exhibits a transition, then it is natural to ask if it satisfies the following stronger property:

There exists a constant $c > 0$ such that for any $\epsilon > 0$, if $M < (c - \epsilon)n$, then $CSP(\mathcal{P})$ is a.s. satisfiable, and if $M > (c + \epsilon)n$, then $CSP(\mathcal{P})$ is a.s. satisfiable.

The current "state-of-the-art" in the analysis of sharp thresholds does not allow us to prove a property this strong even for CNF-satisfiability. Instead, we must weaken the property slightly as follows.

$CSP(\mathcal{P})$ is said to have a *sharp threshold* of satisfiability if there exists a function $c(n)$ bounded away from 0 such that for any $\epsilon > 0$, if $M < (c(n) - \epsilon)n$, then $CSP_{n,M}(\mathcal{P})$ is a.s. satisfiable, and if $M > (c(n) + \epsilon)n$, then $CSP_{n,M}(\mathcal{P})$ is a.s. satisfiable.

Note that the only difference between this and the stronger property is that here we allow c to vary with n. All natural models which are known to have sharp thresholds are conjectured to also satisfy the stronger property where $c(n) = c$ for all n. However, we do not know how to prove this.

This notion of sharp thresholds was introduced (in a broader context than that in the preceding definition) by Friedgut [11], who proved that random instances of k-SAT exhibit a sharp threshold. In doing so, he proved a general theorem characterizing random graph properties which have sharp thresholds. To describe this theorem, we must introduce some definitions.

A graph property, P, is *monotonically increasing* if (i) P is invariant under graph automorphisms, and (ii) whenever H is a subgraph of G, and H has P, then G must have P. This definition extends in the obvious manner to properties of CSPs—condition (ii) becomes the following: if F_1, F_2 are CSPs where every constraint in F_1 is also contained in F_2, and if F_1 has P, then F_2 must have P. The only property which concerns us in this paper is that of being unsatisfiable, which is clearly monotonically increasing.

Given two properties P_1, P_2, their *symmetric difference*, $P_1 \triangle P_2$, is the property of satisfying one but not both of P_1, P_2. Given a set of graphs $\mathcal{H} = \{H_1, \ldots, H_k\}$, we define $Q(\mathcal{H})$ to be the property of having a subgraph which is isomorphic to one of the members of $\mathcal{H}$.

$G_{n,M}$ is the random graph with n vertices and M edges, where each such graph is equally likely to be chosen.

We extend our definitions of *transition* and *sharp threshold* to the setting of random graphs in the obvious manner. A monotonically increasing graph property, P, exhibits a transition if there exist $c_1, c_2 > 0$ such that if $M < c_1 n$, then a.s. $G_{n,M}$ does not have P, and if $M > c_2 n$, then a.s. $G_{n,M}$ has P. A graph property P with a transition has a sharp threshold if there exists a function $c(n)$ bounded away from 0 such that for any $\epsilon > 0$, if $M < (c(n) - \epsilon)n$, then a.s. $G_{n,M}$ does not have P, and if $M > (c(n) + \epsilon)n$, then a.s. $G_{n,M}$ has P.

Friedgut's Theorem yields the following.

FRIEDGUT'S THEOREM (see [11]).[6] *If a monotonically increasing graph property P with a transition does not have a sharp threshold, then for infinitely many values of n there exists some $c(n)$ such that for every $\delta > 0$ there is a set of graphs $\mathcal{H} = \{H_1, \ldots, H_k\}$, each with at most one cycle, such that when $M = c(n) \times n$, the probability of $G_{n,M}$ having $P \triangle Q(\mathcal{H})$ is less than δ. Furthermore, there is a function $B = B(P, \delta)$ such that each H_i has at most B vertices.*

In other words, when $M = c(n)$, P can be arbitrarily closely approximated in probability by the property of having a subgraph from a list of graphs with at most one cycle, where the sizes of these graphs do not grow with n.

This theorem also extends to random hypergraphs and to random CNF-formulae, and Friedgut [10] reports that it extends to the random CSP models discussed here. Thus, roughly speaking, $CSP(\mathcal{P})$ has a sharp threshold of satisfiability iff unsatisfiability cannot be arbitrarily well approximated by containing a subproblem isomorphic to one of a list of problems whose constraint hypergraphs have at most one cycle and whose constraints are drawn from $\mathcal{C}$. If $CSP(\mathcal{P})$ exhibits a transition, then all such

[6] Friedgut's Theorem is in fact much more general than this in that (i) it shows that this is essentially an "if and only if" statement, and (ii) it allows for the possibility of a sharp threshold which occurs when M is not linear in n. We include only this version here for the sake of simplicity and because it covers all situations which are relevant to this paper.

problems are satisfiable. Thus, it would be natural to further expect that $CSP(\mathcal{P})$ has a sharp threshold of satisfiability iff it exhibits a transition. However, this is not the case, as the following example shows.

Example 1. Suppose that $k = 2, d = 4$, and that $\mathcal{C}$ contains exactly two constraints. The first one, C_1, forbids (X_1, X_2) from receiving any pair of values from $\{(1,1),(2,2),(3,3),(4,4)\}$. The second, C_2, forbids any pair (x, y) where one member of the pair is from $\{1,2\}$ and the other member is from $\{3,4\}$. $\mathcal{P}(C_1) = \frac{1}{3}$ and $\mathcal{P}(C_2) = \frac{2}{3}$.

Note that it is easy to verify that $CSP(\mathcal{P})$ exhibits a transition.

Suppose that $M = cn$, where $\frac{3}{4} < c < \frac{3}{2}$. Let G_i be the constraint graph induced by constraints of type C_i. A.s. G_1 will have $\frac{1}{3}M + o(M) = (\frac{1}{2} - c_1)n + o(n)$ edges and G_2 will have $(\frac{1}{2} + c_2)n + o(n)$ edges for some constants $c_1, c_2 > 0$. Therefore, by the most fundamental theorem in random graph theory (see [9]), a.s. no component of G_1 will have size greater than $O(\log n)$, while G_2 will have a *giant component* of size $\alpha n + o(n)$ for a constant $\alpha = \alpha(c) > 0$, and every other component of G_2 will have size at most $O(\log n)$.

Let T be the vertices of the giant component of G_2. Define B to be the event that G_1 has an odd cycle whose vertices all lie in T. It is easy to see that if B holds, then our CSP is unsatisfiable: the C_2 constraints imply that either all variables in T receive 1 or 2, or they all receive 3 or 4. However, if a set of vertices belongs to an odd cycle in G_1, then among them there must be at least 3 values. Furthermore, it is not difficult to verify that the probability that B does not hold and the CSP is unsatisfiable is $o(1)$. Therefore, the probability of satisfiability is $1 - \mathbf{Pr}(B) + o(1)$. It is also easily computed that $\mathbf{Pr}(B)$ tends to a constant (in terms of c) which is strictly between 0 and 1, and so our CSP is neither a.s. satisfiable nor a.s. unsatisfiable—G_1 and G_2 are very close to being two independent random graphs, so this is essentially the probability that a random graph with αn vertices and $(\frac{1}{2} - c_1)n$ edges contains an odd cycle. Therefore this model does not have a sharp threshold despite the fact that it exhibits a transition.

Achlioptas [2] and Friedgut [10] each pointed out that the following set of problems are of the type that Friedgut's Theorem guarantees; i.e., having a subproblem from this set is a good approximation of being unsatisfiable. For each i, let $\mathcal{H}_i$ denote the set of subproblems such that, when G_1, G_2 are defined as above,

(i) G_1 is a cycle of length $2r + 1$ for some positive integer r, with vertices $v_1, \ldots, v_{2r+1}$;
(ii) G_2 consists of r disjoint trees $T_1, \ldots, T_r$ each of size i;
(iii) for each i, v_i is in T_i, and these are the only vertices that G_1, G_2 have in common.

Let P be the property that our random problem is unsatisfiable. For any $\delta > 0$, we can choose i large enough that $\mathbf{Pr}(P \triangle Q(\mathcal{H})) < \delta$. Almost surely, if G_1 has an odd cycle lying in the giant component of G_2, then breadth-first searches, in G_2, from each vertex of the cycle will yield a collection of disjoint trees of length i for any constant i. Therefore, $\mathbf{Pr}(P - Q(\mathcal{H}_i)) = o(1)$. On the other hand, the probability that G_1 has a cycle through the vertex of any nongiant component of G_2 which has size at least i tends to a constant δ_i which tends to 0 as i grows. Choosing i so that $\delta_i < \delta$ yields $\mathbf{Pr}(Q(\mathcal{H}_i)) < \delta$. We omit the details, which are straightforward to a reader who is experienced in random graph theory.

It would be quite interesting to determine some necessary and sufficient conditions on $\mathcal{P}$ for $CSP(\mathcal{P})$ to have a sharp threshold. We close this section by noting the

following sufficient condition.

THEOREM 6. *If*

(a) $\mathcal{C}$ *is very well-behaved;*

(b) *every constraint in* $\mathcal{C}$ *has the property that for each value* δ *and canonical variable* X_i *there is at most one restriction with* $X_i = \delta$*; and*

(c) *for each* $C \in \mathcal{C}$, $\mathcal{P}(C) = \frac{1}{|\mathcal{C}|}$,

then $CSP(\mathcal{P})$ *has a sharp threshold.*

Remark 6. It is not difficult to weaken conditions (b) and (c) somewhat, but (a) is, of course, necessary.

Note that this theorem generalizes the fact that random instances of k-SAT have a sharp threshold for $k \geq 3$ and the fact that k-colorability of random graphs has a sharp threshold for $k \geq 3$. This latter fact was proven by Achlioptas and Friedgut [3], and the proof of Theorem 6 is nearly identical to their proof. To avoid a long repetition of a proof which is readily available, we only give an outline of our proof.

Proof of Theorem 6. Rather than using Friedgut's Theorem, we will make use of a similar theorem proved by Bourgain subsequent to Friedgut's proof. The statement of this theorem is somewhat weaker, but it is easily applied to a more general situation, including the random models considered here.

Consider a set $\mathcal{A}$ of items $\{a_1, \ldots, a_r\}$ and a *selection probability* p. We will choose a random subset $A \subseteq \mathcal{A}$ as follows: for each a_i, we make an independent choice as to whether to include a_i in A, placing it in A with probability p. For example, if $\mathcal{A}$ is the set of possible edges of an n vertex graph, then we are simply choosing the random graph $G_{n,p}$, which is well known (see, for example, [7]) to be in many ways equivalent to $G_{n,M}$, where $M = p\binom{n}{2}$. Under the assumption that $\mathbf{Pr}(C)$ is rational for each $C \in \mathcal{C}$, we can also simulate $CSP(\mathcal{P})$ using this model as follows:

Following Remark 2, we will work in the equivalent model whereby we select each possible hyperedge to be in the constraint hypergraph independently with probability $p = \frac{c \times k!}{n^{k-1}}$.

$\mathcal{A}$ is the set of all $\binom{n}{k} k! |\mathcal{C}|$ possible constraints. We place each constraint into A with selection probability p' (to be named later), and we let our CSP consist of each constraint C such that (i) $C \in A$, and (ii) none of the other $k!|\mathcal{C}| - 1$ possible constraints on the same k-tuple of variables is in A. For any k-tuple of variables, that k-tuple forms a constraint with probability exactly $k!|\mathcal{C}| \times p'(1-p')^{k!|\mathcal{C}|-1}$, and we choose p' so that this value is equal to p. Furthermore, each member of $\mathcal{C}$ is equally likely to be that constraint. Thus, this represents our model precisely. (Note that p' is very close to p—in fact $p' = p(1 + O(n^{-(k-1)}))$.)

This transformation allows us to apply Bourgain's Theorem to $CSP(\mathcal{P})$. As with Friedgut's Theorem, we will not state Bourgain's Theorem in its full power; instead, we will only state its implication to this setting, thus avoiding some technicalities.

BOURGAIN'S THEOREM (see [11]). *Suppose that for each* $c \in \mathcal{C}$, $\mathcal{P}(C) = \frac{1}{|\mathcal{C}|}$. *If* $CSP(\mathcal{P})$ *does not have a sharp threshold of satisfiability, then there exist absolute constants* T, $0 < \alpha < \frac{1}{2}$, *and* $\beta > 0$, *and for infinitely many values of* n *there exists* $c(n)$ *such that when* $M = c(n) \times n$, *the probability that* $CSP(\mathcal{P})$ *is satisfiable is* γ *for some* $\alpha < \gamma < 1 - \alpha$ *and either*

(i) *with probability at least* β, $CSP(\mathcal{P})$ *contains an unsatisfiable subproblem* F_1 *of size at most* T*; or*

(ii) *there exists a satisfiable subproblem* F_1 *of size at most* T *such that conditioning on* $CSP(\mathcal{P})$ *containing* F_1 *lowers the probability of satisfiability to below* $\gamma - \beta$.

In the remainder of this proof, we will only consider values of n from the infinite set of values referred to by Bourgain's Theorem. Because this set is infinite, we can choose arbitrarily high values of n, and so an asymptotic analysis is valid.

A straightforward and well-known[7] random graph argument implies that a.s. every subproblem with at most T constraints is unicyclic. If $\mathcal{C}$ is very well-behaved, all such problems are satisfiable. Thus, (i) does not hold, and so we only need to eliminate the possibility of (ii).

So consider a CSP F_1 as in (ii). Without loss of generality, we can assume that F_1 is on the variables $x_1, \ldots, x_t$ for some $t \leq T$. Consider any satisfying assignment $x_1 = \delta_1, \ldots, x_t = \delta_t$ for F_1. The probability that $CSP(\mathcal{P})$ is satisfiable, conditional on $CSP(\mathcal{P})$ containing F_1, is at most the probability that $CSP(\mathcal{P})$ has a satisfying assignment with $x_1 = \delta_1, \ldots, x_t = \delta_t$.

Let $\mathcal{D}$ be the set of constraints, other than those in F_1, which involve $x_1, \ldots, x_t$. A straightforward argument shows that there is an absolute constant R such that with probability at least $\frac{1-\beta}{2}$ (a) $|\mathcal{D}| \leq R$, (b) no constraint in $\mathcal{D}$ involves two variables from $\{x_1, \ldots, x_t\}$, and (c) no variable outside of $\{x_1, \ldots, x_t\}$ lies in more than one constraint in $\mathcal{D}$.

Consider any constraint $D \in \mathcal{D}$, say on variables $x_i, x_{j_1}, \ldots, x_{j_{k-1}}$ where $1 \leq i \leq t$ and each other index is greater than t. Since each constraint in $\mathcal{C}$ contains at most one restriction involving $x_i = \delta_i$, insisting that $x_i = \delta_i$ at worst reduces to imposing a single restriction of size $k-1$ on $x_{j_1}, \ldots, x_{j_{k-1}}$. This is no more restrictive than forbidding a single value for each x_{j_l}. Thus, conditional on (a), (b), and (c) holding, the probability that $CSP(\mathcal{P})$ has a satisfying assignment with $x_1 = \delta_1, \ldots, x_t = \delta_t$ is bounded from below by the following experiment:

Fix some collection of values $\zeta_1, \ldots, \zeta_{R\times(k-1)}$. Choose a random CSP from $CSP(\mathcal{P})$ with $M = c(n)$. Pick $R \times (k-1)$ variables at random and add the additional requirements that the ith variable cannot receive ζ_i.

Since the probability that either (a), (b), or (c) fails is at most $\frac{\beta}{2}$, condition (ii) implies that these additional requirements do not decrease the probability of satisfiability by more than $\frac{\beta}{2}$. We will show that this is impossible. To do so requires two steps:

(1) We show that if adding these $R\times(k-1)$ additional requirements decreases the probability of satisfiability by at least $\frac{\beta}{2}$, then so would increasing M from $c(n) \times n$ to $c(n) \times n + R^{'}$ for an absolute constant $R^{'}$ defined in terms of $R \times (k-1)$.

(2) Increasing M from $c(n)\times n$ to $c(n)\times n + R^{'}$ will only decrease the probability of unsatisfiability by $o(1)$.

Step (2) is at least intuitively obvious: adding a relatively very small number of constraints should not have a significant effect on the probability of satisfiability. However, it takes a little work to prove it.

The proof of step (1) goes as follows: Suppose that we were adding only one additional requirement: we pick a random variable and forbid it from receiving ζ_1. Let X_1 be the set of variables such that, before this requirement, the only satisfying assignments set all of X_1 equal to ζ_1. If this requirement causes our problem to turn from satisfiable to unsatisfiable, then we must have chosen a random variable in X_1. Thus, in order for this to have a significant probability of happening, X_1 would have to contain at least αn variables for some constant $\alpha > 0$. Suppose that this is the case. Then if instead of adding this extra requirement, we added a single additional

[7] See [7] for a proof of the case $k = 2$. The proof for higher values of k is similar.

constraint, we would cause the CSP to become unsatisfiable with probability at least $\alpha^k \times \frac{1}{|\mathcal{C}|}$, which is a positive constant. The reason is that this is the probability that the entire k-tuple is from X_1 and that we choose the restriction which forbids them from all being assigned the value ζ_1. Thus adding an additional constraint has a serious effect on the probability of satisfiability.

Modifying this argument to the case where we have $R(k-1)$ additional requirements uses the same ideas but is technically a little complicated.

The proofs of steps (1) and (2) are identical to the corresponding steps found in [3], [11], and [1]. We refer the reader to any of these for more details. □

Acknowledgment. I'm grateful to Ehud Friedgut for his many helpful comments and corrections.

REFERENCES

[1] D. Achlioptas, *Threshold Phenomena in Random Graph Colouring and Satisfiability*, Ph.D. Thesis, Department of Computer Science, University of Toronto, 1999.
[2] D. Achlioptas, *personal communication*.
[3] D. Achlioptas and E. Friedgut, *A threshold for k-colourability*, Random Structures Algorithms, 14 (1999), pp. 63–70.
[4] D. Achlioptas, L. Kirousis, E. Kranakis, and D. Krizanc, *Rigorous results for random* $(2+p)$*-SAT*, in Proceedings of RALCOM '97, Santorini, Greece, 1997, pp. 14–23.
[5] D. Achlioptas, L. Kirousis, E. Kranakis, D. Krizanc, M. Molloy, and Y. Stamatiou, *Random constraint satisfaction: A more accurate picture*, Constraints, 6 (2001), pp. 329–344.
[6] N. Alon and J. Spencer, *The Probabilistic Method*, Wiley, New York, 1992.
[7] B. Bollobás, *Random Graphs*, Academic Press, New York, 1985.
[8] N. Creignou and H. Daudé, *Random generalized satisfiability problems*, in Proceedings of the 5th International Symposium on the Theory and Applications of Satisfiability Testing, Cincinnati, OH, 2002, pp. 17–26.
[9] P. Erdős and A. Rényi, *On the evolution of random graphs*, Publication of the Mathematical Institute of the Hungarian Academy of Sciences, 5 (1960), pp. 17–61.
[10] E. Friedgut, *personal communication*.
[11] E. Friedgut, *Sharp thresholds of graph properties and the k-SAT problem*, with an appendix by J. Bourgain, J. Amer. Math. Soc., 12 (1999), pp. 1017–1054.
[12] I. Gent, E. MacIntyre, P. Prosser, B. Smith, and T. Walsh, *Random constraint satisfaction: Flaws and structure*, Constraints, 6 (2001), pp. 345–372.
[13] R. Karp, *The transitive closure of a random digraph*, Random Structures Algorithms, 1 (1990), pp. 73–93.
[14] M. Molloy, *When does the giant component bring unsatisfiability?*, in preparation.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 950–1006

MACRO TREE TRANSLATIONS OF LINEAR SIZE INCREASE ARE MSO DEFINABLE*

JOOST ENGELFRIET† AND SEBASTIAN MANETH†‡

Abstract. The first main result is that if a macro tree translation is of linear size increase, i.e., if the size of every output tree is linearly bounded by the size of the corresponding input tree, then the translation is MSO definable (i.e., definable in monadic second-order logic). This gives a new characterization of the MSO definable tree translations in terms of macro tree transducers: they are exactly the macro tree translations of linear size increase. The second main result is that given a macro tree transducer, it can be decided whether or not its translation is MSO definable, and if it is, then an equivalent MSO transducer can be constructed. Similar results hold for attribute grammars, which define a subclass of the macro tree translations.

Key words. tree transducers, monadic second-order logic, decidability

AMS subject classifications. 68Q45, 68Q42, 03D05

DOI. 10.1137/S0097539701394511

1. Introduction.

Very often a complex object has a structure that shows how it is composed from smaller objects by the application of certain operations. The smaller objects may themselves be composed of other objects. Such a structure can naturally be described as a tree, and hence the objects are "tree-structured." Examples of tree-structured objects are the words of a context-free language (with derivation trees as structure) or the graphs of bounded tree-width (with tree decompositions as structure). Now consider the transformation of a tree-structured object, based on its structure and independent of the interpretation of the operations, i.e., a tree-to-tree transformation. We are interested in models of such transformations: tree transducers. Well-known examples of tree transducers are top-down tree transducers [46, 48, 1, 32] and attribute grammars [17, 27, 28] (motivated by syntax-directed semantics and compilers; cf. [35, 37, 40, 53]), unranked tree transducers [43, 2] and pebble tree transducers [45] (motivated by the transformation of XML documents; cf. [51]), and macro tree transducers [15, 8, 9, 24, 28] (motivated by syntax-directed and denotational semantics [35, 47], and used as a model in, e.g., functional programming [52, 39, 41], language prototyping [49], and linguistics [38, 44]). Motivated by model theory is the idea of "interpretation," meaning the definition of a (logical) structure in terms of logical formulas over another structure (cf. Chapter 10 of [13]). For monadic second-order (MSO) logic, such MSO interpretations have recently been used to characterize the generation of graphs by context-free graph grammars [5, 7, 23, 16] (see also [36]). Taking trees as a logical structure, another type of tree transducer is obtained: the MSO tree transducer, studied in [3, 19] (for strings, see [18]). An important part of tree transducer theory is comparing the formal power of these different models of transformation of tree-structured objects and providing effective translations between these models. This paper compares the power of macro tree transducers and MSO tree transducers.

*Received by the editors August 30, 2001; accepted for publication (in revised form) March 12, 2003; published electronically June 25, 2003.
http://www.siam.org/journals/sicomp/32-4/39451.html
†Leiden University, LIACS, P.O. Box 9512, 2300 RA Leiden, The Netherlands (engelfri@liacs.nl).
‡Present address: Swiss Institute of Technology Lausanne, Programming Methods Laboratory (LAMP), 1015 Lausanne, Switzerland (sebastian.maneth@epfl.ch).

The macro tree transducer (MTT) is a finite state device that translates, in a recursive top-down fashion, an input tree into an output tree, handling context information by the use of parameters. The states of the MTT can be viewed as functions that call each other recursively; the initial state is the main function. The (tree-to-tree) translations of MTTs form a large class containing the translations of top-down tree transducers and attribute grammars. In order to prove our results, we add the feature of regular look-ahead (see, e.g., section 18 of [32]) to top-down tree transducers, attribute grammars, and MTTs. Note that in the case of MTTs this has no influence on the translations: the classes of translations realized by MTTs with and without regular look-ahead are the same [24].

The MSO tree transducer uses formulas in monadic second-order logic to define tree-to-tree translations. This provides a declarative way of defining a tree translation, as opposed to the operational way of an MTT. The idea is to define the nodes and edges of the output tree in terms of MSO formulas that are interpreted in the input tree, or, more precisely, in a fixed number of disjoint copies of the input tree. Tree translations definable in MSO logic have nice properties, comparable to those of finite state transductions on strings. In particular, they are closed under composition and they can be computed in linear time. Macro tree translations do not possess these properties.

The question arises, What is the precise relationship between these two different models? From [3, 19] it is known that every MSO definable tree translation can be realized by an MTT. However, the converse does not hold for obvious reasons: by definition, MSO definable tree translations are of linear size increase: the size of the output tree is at most k times the size of the input tree, where k is the number of disjoint copies of the input tree, used to define the output tree. On the other hand, the translations realized by MTTs can be of double exponential size increase (cf. Lemma 4.22 of [28]). Our first main result is that if we restrict ourselves to translations of linear size increase, then the two formalisms, MSO tree transducers and MTTs, have exactly the same power, i.e., the respective classes of translations coincide.

Let us briefly discuss the proof of the first main result. As mentioned before, our MTTs are equipped with regular look-ahead. In [19] a characterization of the MSO definable tree translations in terms of MTTs is given: they are the translations realized by "finite copying" MTTs. The notion of finite copying was introduced in [1] for generalized syntax-directed translation schemes, which are closely related to top-down tree transducers. It requires that there be a bound on the number of occurrences of states that translate a given node of the input tree. For MTTs this requirement is called "finite copying in the input," and an MTT is finite copying [19] if it is both finite copying in the input and "finite copying in the parameters"; the latter means that there is a bound on the number of copies made of a parameter. We want to prove that if the translation realized by an MTT is of linear size increase, then it is MSO definable. By the above this is equivalent to showing that for every MTT M that is of linear size increase (i.e., which realizes a translation of linear size increase), there is an equivalent MTT M' that is finite copying. How can we construct M', given M? The idea is that every MTT M can be transformed into a normal form M', called the "proper normal form" of M, such that if M is of linear size increase, then M' is finite copying. Roughly speaking this normal form requires that all states and parameters of M' are really "needed"; more precisely, each state generates infinitely many output trees (considering all possible input trees), and for

each parameter y there are infinitely many actual parameter trees being substituted for y (for all possible input trees). Then for a proper MTT M' it can be shown that (i) if M' is of linear size increase, then it is finite copying in the parameters, and (ii) if M' is finite copying in the parameters and of linear size increase, then it is finite copying in the input. Both (i) and (ii) are proved by a pumping argument; i.e., it is shown that if M' is not finite copying in the parameters, then it is not of linear size increase, and similarly for (ii).

Our second main result concerns decidability. Given an MTT it can be decided whether or not its translation is MSO definable, and if so, an equivalent MSO tree transducer can be constructed. The proof is based on the following results: (1) the translation realized by an MTT M is MSO definable—i.e., of linear size increase—if and only if its proper normal form M' is finite copying (by the proof of our first main result, as discussed above); (2) for an MTT it is decidable whether or not it is finite copying (the proof is based on the fact that the finiteness of ranges of MTTs is decidable [12]); and (3) from [19, 3] it follows that given a finite copying MTT, an equivalent MSO tree transducer can be constructed.

Note that very often membership in a subclass is undecidable (such as regularity of a context-free language). In cases of decidability there is often a characterization of the subclass that is independent of the device that defines the whole class, i.e., a "semantic" rather than "syntactic" characterization, such as our linear size increase characterization. As another example, in [6] it is shown that an NR (node replacement) context-free graph language can be generated by an HR (hyperedge replacement) context-free graph grammar if and only if the number of edges of its graphs is linearly bounded by the number of nodes.

The idea for our main results stems from [1]; there it is shown that a generalized syntax-directed translation (gsdt) scheme can be realized by a tree-walking transducer if and only if it is of linear size increase. Since gsdt schemes are a variation of top-down tree transducers, and tree-walking transducers are closely related to finite copying top-down tree transducers [22], our result can be viewed as a generalization of the result of [1], from top-down tree transducers to MTTs. In fact, since the proper normal form of a top-down tree transducer is again a top-down tree transducer, we reobtain their result (in our formalism): the top-down tree translations of linear size increase are exactly the translations realized by finite copying top-down tree transducers. Moreover, they are exactly the MSO definable top-down tree translations.

The main result of [19], on which this paper is based, is in turn based on the main result of [3], which states that the MSO definable tree translations can be characterized by attribute grammars (more precisely, by attributed tree transducers with look-ahead) that are single-use restricted. The single-use restriction [33, 30, 39, 41] is interesting, because attribute grammars are closed under left-composition with single-use restricted attribute grammars. Our results now imply that given an attributed tree transducer (with look-ahead) it can be decided whether or not there exists an equivalent one that is single-use restricted, and furthermore that the linear size increase attributed tree translations are precisely the MSO definable tree translations.

This paper is structured as follows. In section 2 trees and tree substitutions are defined. In particular, the definition of second-order tree substitution is given, which is the type of substitution that MTTs are based on. Various results about these substitutions are proved, for example, how to compute the number of occurrences of a particular symbol in a tree to which a second-order tree substitution is applied. Then tree languages and tree translations are defined, and the notion of MSO definable tree

translation is recalled briefly. Section 3 defines MTTs, which are total deterministic and equipped with regular look-ahead. Some basic results needed in the paper are recalled, and two subclasses defined by restrictions on the parameters are considered. Section 4 recalls the notion of finite copying, which consists of two parts: finite copying in the input and finite copying in the parameters. It is proved that it is decidable for an MTT whether or not it is finite copying. Moreover, although this is already known from the result of [19], it is proved for the sake of completeness that if an MTT is finite copying, then it is of linear size increase. The proof is based on an intermediate, very natural notion of bounded copying: "finite contribution." An MTT is of finite contribution if there is a bound on the number of output nodes that are contributed by a given node of the input tree. Also in this section the notion of "finite nested copying in the input" is introduced; it requires a bound on the amount of nesting of the states that translate a given node of the input tree. In section 5 the proper normal form is introduced, and it is shown how to construct, given an MTT, an equivalent one in proper normal form. Section 6 proves our main results: if the translation realized by a proper MTT M is of linear size increase (for short, "M is lsi"), then M is finite copying. The proof goes in three stages: (I) If M is lsi, then it is finite nested copying in the input. (II) If M is lsi and finite nested copying in the input, then it is finite copying in the parameters. And finally, (III) if M is lsi, finite nested copying in the input, and finite copying in the parameters, then it is finite copying in the input. Section 7 presents the main results and their consequences for top-down tree transducers, attribute grammars, and context-free graph grammars. At last, some open problems and further research topics are mentioned.

We note that technically this paper is concerned with MTTs only. The links to MSO tree transducers were established in [3, 19].

2. Preliminaries. The set $\{0, 1, \dots\}$ of natural numbers is denoted by $\mathbb{N}$. The empty set is denoted by $\varnothing$. For $k \in \mathbb{N}$, $[k]$ denotes the set $\{1, \dots, k\}$; thus $[0] = \varnothing$. For a set A, $|A|$ is the cardinality of A, and A^* is the set of all strings over A. The empty string is denoted by ε. The length of a string w is denoted $|w|$, and the number of occurrences of the symbol a in w is denoted by $\#_a(w)$. For a set $B \subseteq A$, $\#_B(w) = \sum\{\#_a(w) \mid a \in B\}$. For strings $v, w_1, \dots, w_n \in A^*$ and distinct $a_1, \dots, a_n \in A$, we denote by $v[a_1 \leftarrow w_1, \dots, a_n \leftarrow w_n]$ the result of (simultaneously) substituting w_i for every occurrence of a_i in v. Note that the substitution $[a_1 \leftarrow w_1, \dots, a_n \leftarrow w_n]$ is a homomorphism on strings. Let P be a condition on a and w such that $\{(a, w) \mid P\}$ is a partial function; then we use, similar to set notation, $[a \leftarrow w \mid P]$ to denote the substitution $[L]$, where L is the list of all $a \leftarrow w$ for which condition P holds.

2.1. Trees. A set Σ together with a mapping $\mathrm{rank}_\Sigma\colon \Sigma \to \mathbb{N}$ is called a *ranked set*. For $k \geq 0$, $\Sigma^{(k)}$ is the set $\{\sigma \in \Sigma \mid \mathrm{rank}_\Sigma(\sigma) = k\}$; we also write $\sigma^{(k)}$ to indicate that $\mathrm{rank}_\Sigma(\sigma) = k$. For sets Σ and A, $\langle \Sigma, A\rangle = \Sigma \times A$; if Σ is ranked, then so is $\langle \Sigma, A\rangle$, with $\mathrm{rank}_{\langle \Sigma, A\rangle}(\langle \sigma, a\rangle) = \mathrm{rank}_\Sigma(\sigma)$ for every $\langle \sigma, a\rangle \in \langle \Sigma, A\rangle$. A *ranked alphabet* is a finite ranked set.

For the rest of this paper we choose the *set of input variables* to be $X = \{x_1, x_2, \dots\}$ and the *set of parameters* to be $Y = \{y_1, y_2, \dots\}$. For $k \geq 0$, $X_k = \{x_1, \dots, x_k\}$ and $Y_k = \{y_1, \dots, y_k\}$. In this paper (as opposed to other papers) we allow a ranked set to contain parameters. However, by convention, if a parameter is an element of a ranked set Σ, then we require that it have rank zero; i.e., we require that $\Sigma \cap Y \subseteq \Sigma^{(0)}$. Thus, *parameters have rank zero.*

Let Σ be a ranked set. The *set of trees over* Σ, denoted by T_Σ, is the smallest set of strings $T \subseteq \Sigma^*$ such that if $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $t_1, \dots, t_k \in T$, then $\sigma\, t_1 \cdots t_k \in T$.

For better readability we will usually write $\sigma(t_1, \ldots, t_k)$ for $\sigma\, t_1 \cdots t_k$. For a set of parameters $Y' \subseteq Y$ we will also use $T_\Sigma(Y')$ to denote the set of trees $T_{\Sigma \cup Y'}$, where $\Sigma \cup Y'$ is the ranked set with $\mathrm{rank}_{\Sigma\cup Y'}(\sigma) = \mathrm{rank}_\Sigma(\sigma)$ for $\sigma \in \Sigma$, and $\mathrm{rank}_{\Sigma\cup Y'}(y) = 0$ for $y \in Y'$.

For every tree $t \in T_\Sigma$, the *set of nodes of* t, denoted by $V(t)$, is a subset of $\mathbb{N}^*$ which is inductively defined as follows: if $t = \sigma(t_1, \ldots, t_k)$ with $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and for all $i \in [k], t_i \in T_\Sigma$, then $V(t) = \{\varepsilon\} \cup \{iu \mid u \in V(t_i), i \in [k]\}$. Thus, ε represents the root of a tree, and for a node u the ith child of u is represented by ui. A *leaf* is a node without children. If $u = vw$ with $w \in \mathbb{N}^*$, then v is an *ancestor of* u and u is a *descendant of* v; if $w \neq \varepsilon$, then v is a *proper ancestor of* u, and u is a *proper descendant of* v. The *label of* t *at node* u is denoted by $t[u]$; we also say that $t[u]$ occurs in t (at u). The *subtree of* t *at node* u is denoted by t/u. The *substitution of the tree* $s \in T_\Sigma$ *at node* u *in* t is denoted by $t[u \leftarrow s]$; it means that the subtree t/u is replaced by s. Formally, these notions can be defined as follows: $t[\varepsilon]$ is the first symbol of t (in Σ), $t/\varepsilon = t$, $t[\varepsilon \leftarrow s] = s$, and if $t = \sigma(t_1, \ldots, t_k)$, $i \in [k]$, and $u \in V(t_i)$, then $t[iu] = t_i[u]$, $t/iu = t_i/u$, and $t[iu \leftarrow s] = \sigma(t_1, \ldots, t_i[u \leftarrow s], \ldots, t_k)$.

The usual preorder of the nodes of t (which, in fact, is the lexicographical order on $\mathbb{N}^*$) is denoted $<$; thus, $\varepsilon < iu$ (for $i \geq 1$), if $u < v$, then $iu < iv$, and if $i < j$, then $iu < jv$.

The *size* of a tree t, denoted by $\mathrm{size}(t)$, is the number $|V(t)|$ of nodes of t. For $t = \sigma(t_1, \ldots, t_k)$, $\mathrm{size}(t)$ equals $1 + \mathrm{size}(t_1) + \cdots + \mathrm{size}(t_k)$; note that $\mathrm{size}(t) = \sum_{\sigma\in\Sigma} \#_\sigma(t) = |t|$. For $\sigma \in \Sigma$, $V_\sigma(t)$ denotes the set of nodes of t which are labeled by σ, i.e., $\{u \in V(t) \mid t[u] = \sigma\}$; note that $|V_\sigma(t)| = \#_\sigma(t)$: the number of occurrences of σ in t. For a set $S \subseteq \Sigma$, $V_S(t) = \bigcup_{\sigma\in S} V_\sigma(t)$. The *height* of t is denoted by $\mathrm{height}(t)$; for $t = \sigma(t_1, \ldots, t_k)$ it equals $1 + \max\{\mathrm{height}(t_i) \mid i \in [k]\}$.

2.2. Tree substitution. In the previous subsection on trees we already defined a particular tree substitution: for trees t, s and a node u of t, $t[u \leftarrow s]$ is the result of replacing in t the subtree t/u by s. Now we want to consider replacing in t all occurrences of a symbol σ.

Trees are particular strings and therefore string substitution as defined in the beginning of these preliminaries is applicable to a tree. In order to guarantee that the resulting string is again a tree, we require that only symbols of rank zero, i.e., leaves, may be replaced by trees; we refer to this type of substitution as "first-order tree substitution." Note that top-down tree transducers are based on first-order tree substitution. In contrast to this, "second-order tree substitution" means that symbols of arbitrary rank can be replaced by a tree. This is the type of substitution MTTs are based on. Consider the replacement of a symbol σ of rank k by a tree s in which the parameter symbols $y_1, \ldots, y_k$ occur to indicate where the subtrees of σ have to be inserted. Now, if σ occurs at a node u of the tree t, then replacing it by s means to replace the subtree t/u of t by the tree $s[y_1 \leftarrow t/u1, \ldots, y_k \leftarrow t/uk]$. (Hence the first-order tree substitution is used to define the second-order one.) Now we define second-order tree substitution formally. Since all occurrences of σ have to be replaced simultaneously, an inductive definition is appropriate.

Let Σ be a ranked set and let $\sigma_1, \ldots, \sigma_n$ be distinct elements of $\Sigma - Y$, $n \geq 1$, and for each $i \in [n]$ let s_i be a tree in $T_{\Sigma-Y}(Y_{\mathrm{rank}_\Sigma(\sigma_i)})$. For $t \in T_\Sigma$, the *second-order tree substitution of* σ_i *by* s_i *in* t, denoted by $t[\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$, is inductively defined as follows (abbreviating $[\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ by $[\![\ldots]\!]$): For $t = \sigma(t_1, \ldots, t_k)$ with $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $t_1, \ldots, t_k \in T_\Sigma$, (i) if $\sigma = \sigma_i$ for an $i \in [n]$, then $t[\![\ldots]\!] = s_i[y_j \leftarrow t_j[\![\ldots]\!] \mid j \in [k]]$ and (ii) otherwise $t[\![\ldots]\!] = \sigma(t_1[\![\ldots]\!], \ldots, t_k[\![\ldots]\!])$.

We will say that $[\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ is a second-order tree substitution over Σ. Note that it is a mapping from T_Σ to T_Σ. In fact, it is a tree homomorphism [31]. Note also that (just as first-order tree substitution) second-order tree substitution is associative (by the closure of tree homomorphisms under composition; cf. Theorem IV.3.7 of [31]), i.e., $t[\![\sigma \leftarrow s]\!][\![\sigma \leftarrow s']\!] = t[\![\sigma \leftarrow s[\![\sigma \leftarrow s']\!]]\!]$, and if $\sigma' \neq \sigma$, then $t[\![\sigma \leftarrow s]\!][\![\sigma' \leftarrow s']\!] = t[\![\sigma' \leftarrow s', \sigma \leftarrow s[\![\sigma' \leftarrow s']\!]]\!]$, and similarly for the general case (cf. sections 3.4 and 3.7 of [4]). Let P be a condition on σ and s such that $\{(\sigma, s) \mid P\}$ is a partial function; then we use $[\![\sigma \leftarrow s \mid P]\!]$ to denote the substitution $[\![L]\!]$, where L is the list of all $\sigma \leftarrow s$ for which condition P holds. In second-order tree substitutions we use for the relabeling $\sigma \leftarrow \delta(y_1, \ldots, y_k)$ of $\sigma^{(k)}$ by $\delta^{(k)}$ the abbreviation $\sigma \leftarrow \delta$; note that this is, in fact, a string substitution.

Note that the restrictions on σ_i and s_i in the first sentence of the previous paragraph are rather subtle. Recall from section 2.1 that Σ can contain parameters. However, for technical reasons, we do not want a second-order tree substitution to replace parameters (hence $\sigma_i \in \Sigma - Y$), and we do not want it to introduce parameters (hence $s_i \in T_{\Sigma - Y}(Y_{\mathrm{rank}_\Sigma(\sigma_i)})$, which means that $s_i \in T_{\Sigma \cup Y}$, and if y_j occurs in s_i, then $j \leq \mathrm{rank}_\Sigma(\sigma_i)$).

The second-order tree substitution $[\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ is *nondeleting* if for every $i \in [n]$, $\#_{y_j}(s_i) \geq 1$ for all $j \in [\mathrm{rank}_\Sigma(\sigma_i)]$, and it is *nonerasing* if for every $i \in [n]$, $s_i \notin Y$. It is *productive* if it is both nondeleting and nonerasing.

LEMMA 2.1. *Let Σ be a ranked alphabet and $\Phi = [\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ a nondeleting second-order tree substitution over Σ. For all $t, t' \in T_\Sigma$, if t' is a subtree of t, then $t'\Phi$ is a subtree of $t\Phi$. In particular, for $y \in Y$, if $\#_y(t) \geq 1$, then $\#_y(t\Phi) \geq 1$.*

Proof. For $t = \sigma(t_1, \ldots, t_k)$, $t_j\Phi$ is a subtree of $t\Phi$. Hence the result follows immediately by induction on the structure of t.

If $\#_y(t) \geq 1$, then y is a subtree of t (because, by convention, the parameter y has rank 0). This means, by the first part of this lemma, that y is also a subtree of $t\Phi$, i.e., $\#_y(t\Phi) \geq 1$. Note that $y\Phi = y$ because, by the definition of second-order tree substitution, $\sigma_i \notin Y$ for all $i \in [n]$. □

LEMMA 2.2. *Let Σ be a ranked alphabet and $\Phi = [\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ a nonerasing second-order tree substitution over Σ. For every $t \in T_\Sigma$, if $t \notin Y$, then $t\Phi \notin Y$.*

Proof. Let $t = \sigma(t_1, \ldots, t_k)$ with $\sigma \in \Sigma^{(k)} - Y$. If $\sigma \notin \{\sigma_1, \ldots, \sigma_n\}$, then $t\Phi = \sigma(t_1\Phi, \ldots, t_k\Phi) \notin Y$. If $\sigma = \sigma_i$ for some $i \in [n]$, then $t\Phi = s_i[y_j \leftarrow t_j\Phi \mid j \in [k]] \notin Y$ (because $s_i \notin Y$). □

In order to calculate the number of times that a particular node u of a tree is copied by the application of a second-order tree substitution, we need to know which symbols occur at the ancestors of u. For this we define the string obtained by reading the labels of the ancestors of u in descending order, starting at the root; if u is labeled by a parameter, then we do not include its label in this string, because in trees of the form $t[\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ the parameters present in the trees s_i do not appear (by the requirement that $s_i \in T_{\Sigma - Y}(Y_{\mathrm{rank}_\Sigma(\sigma_i)})$).

For a tree $t \in T_\Sigma$ and a node $u \in V(t)$, the *label path to u (in t)*, denoted by $\mathrm{lpath}(t, u)$, is the string in $(\Sigma - Y)^*$ defined recursively as follows: $\mathrm{lpath}(t, \varepsilon) = \varepsilon$ if $t \in Y$, and otherwise $\mathrm{lpath}(t, \varepsilon) = t[\varepsilon]$; for $i \geq 1$ and $u \in \mathbb{N}^*$, $\mathrm{lpath}(t, iu) = t[\varepsilon]\,\mathrm{lpath}(t/i, u)$. For example, let t be the tree $\gamma(\sigma(a, y_1))$; then $\mathrm{lpath}(t, 12) = \gamma\,\mathrm{lpath}(\sigma(a, y_1), 2) = \gamma\sigma\,\mathrm{lpath}(y_1, \varepsilon) = \gamma\sigma$, $\mathrm{lpath}(t, 1) = \gamma\sigma$, and $\mathrm{lpath}(t, 11) = \gamma\sigma a$.

The following lemma shows how a label path in t changes if a second-order tree

substitution is applied to t.

LEMMA 2.3. *Let Σ be a ranked alphabet, Φ the second-order tree substitution $[\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ over Σ, and $t \in T_\Sigma$.*

(i) *Every label path in $t\Phi$ is of the form $w_0 v_1 w_1 \cdots v_m w_m$, where $m \geq 0$, $w_0 \sigma_{i_1} w_1 \cdots \sigma_{i_m} w_m$ is a label path in t, $i_1, \ldots, i_m \in [n]$, v_j is a label path in s_{i_j} for $j \in [m]$, and $w_0, \ldots, w_m \in (\Sigma - \{\sigma_1, \ldots, \sigma_n\})^*$.*

(ii) *If Φ is nondeleting, then for every $w, v \in \Sigma^*$ such that $w\sigma_i$ is a label path in t and v is a label path in s_i, there is a $w' \in \Sigma^*$ such that $w'v$ is a label path in $t\Phi$.*

2.3. Number of occurrences. Since this paper is about the size increase of MTTs, and they are based on second-order tree substitution, we need to know how the size of a tree t changes when a second-order tree substitution Φ is applied to t. Recall that $\mathrm{size}(t\Phi)$ is the sum of the numbers $\#_\sigma(t\Phi)$ of occurrences of σ in $t\Phi$ for all symbols σ. Thus, we need to determine the number $\#_\sigma(t\Phi)$. Since second-order tree substitution is based on first-order tree substitution which is a particular string substitution, we first determine the number $\#_a(w[\ldots])$, where w is a string and $[\ldots]$ is a string substitution.

The following lemma can be proved by straightforward induction on the length of w.

LEMMA 2.4. *Let A be an alphabet. Let $w, v_1, \ldots, v_n \in A^*$ and let $a_1, \ldots, a_n$ be distinct elements of A. For every $a \in A$,*

$$\#_a(w[a_1 \leftarrow v_1, \ldots, a_n \leftarrow v_n]) = S^a + \sum_{i \in [n]} \#_{a_i}(w) \cdot \#_a(v_i),$$

where $S^a = \#_a(w)$ if $a \notin \{a_1, \ldots, a_n\}$, and otherwise $S^a = 0$.

In the next lemma we prove the generalization of Lemma 2.4 to second-order tree substitution. Intuitively we now have to take into account, for a node u of the tree t, how many times it is copied by the application of the second-order tree substitution $\Phi = [\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$. For each σ_i that occurs at a proper ancestor u' of u, u is in some subtree $t/u'j$ of u'; thus, replacing σ_i by s_i generates $\#_{y_j}(s_i)$ copies of $t/u'j$. Hence, the product of these numbers $\#_{y_j}(s_i)$, for all proper ancestors u', determines the number of copies of u in $t\Phi$. In the lemma this product is denoted $\prod F^\Phi_{t,u}$, where the family $F^\Phi_{t,u}$ of numbers is defined as follows.

DEFINITION 2.5 (the family $F^\Phi_{t,u}$). *Let Σ be a ranked alphabet and $\Phi = [\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ a second-order tree substitution over Σ. For every $t \in T_\Sigma$ and $u \in V(t)$, define the family $F^\Phi_{t,u}$ as*

$$F^\Phi_{t,u} = \{f_{u'}\}_{u' \text{ proper ancestor of } u},$$

where

$$f_{u'} = \begin{cases} 1 & \text{if } t[u'] \notin \{\sigma_1, \ldots, \sigma_n\}, \\ \#_{y_j}(s_i) & \text{if } t[u'] = \sigma_i, i \in [n], \text{ and } u = u'ju'' \text{ with } j \geq 1, u'' \in \mathbb{N}^*. \end{cases}$$

Note that if $u = \varepsilon$, i.e., $F^\Phi_{t,u}$ is empty, then $\prod F^\Phi_{t,u} = 1$.

LEMMA 2.6. *Let Σ be a ranked alphabet, $\Phi = [\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ a second-order tree substitution over Σ, and $t \in T_\Sigma$. For every $\sigma \in \Sigma$,*

$$\#_\sigma(t\Phi) = S^\sigma_1 + S^\sigma_2,$$

where

$$S_1^\sigma = \sum_{u \in V_\sigma(t)} \prod F_{t,u}^\Phi \quad \textit{if } \sigma \notin \{\sigma_1, \ldots, \sigma_n\}, \textit{ and otherwise } S_1^\sigma = 0,$$

$$S_2^\sigma = \sum_{u \in V_{\sigma_i}(t), i \in [n]} \#_\sigma(s_i) \cdot \prod F_{t,u}^\Phi \quad \textit{if } \sigma \notin Y, \textit{ and otherwise } S_2^\sigma = 0.$$

Proof. Denote $\{\sigma_1, \ldots, \sigma_n\}$ by Σ_n. Let $O_\varepsilon = V_\sigma(t) \cap \{\varepsilon\}$, $O = V_\sigma(t) - \{\varepsilon\}$, and for $i \in [n]$, $O_{\varepsilon,i} = V_{\sigma_i}(t) \cap \{\varepsilon\}$ and $O_i = V_{\sigma_i}(t) - \{\varepsilon\}$. Clearly, $S_1^\sigma = T_1 + S_1$, where for $\sigma \notin \Sigma_n$, $T_1 = \sum_{u \in O_\varepsilon} \prod F_{t,u}^\Phi$ and $S_1 = \sum_{u \in O} \prod F_{t,u}^\Phi$, and otherwise $T_1 = 0$ and $S_1 = 0$. Similarly, $S_2^\sigma = T_2 + S_2$, where for $\sigma \notin Y$, $T_2 = \sum_{u \in O_{\varepsilon,i}, i \in [n]} \#_\sigma(s_i) \cdot \prod F_{t,u}^\Phi$ and $S_2 = \sum_{u \in O_i, i \in [n]} \#_\sigma(s_i) \cdot \prod F_{t,u}^\Phi$, and otherwise $T_2 = 0$ and $S_2 = 0$.

The proof that $S_1^\sigma + S_2^\sigma$ equals $\#_\sigma(t\Phi)$ is by induction on the structure of t. Let $t = \sigma'(t_1, \ldots, t_k)$ with $\sigma' \in \Sigma^{(k)}$, $k \geq 0$, and $t_1, \ldots, t_k \in T_\Sigma$.

Case 1. $\sigma' \in \Sigma - \Sigma_n$.

Then $t[\varepsilon] \notin \Sigma_n$ and hence, for every $j \in [k]$ and $v \in V(t_j)$, $\prod F_{t,jv}^\Phi = \prod F_{t_j,v}^\Phi$. Since O equals $\bigcup_{j \in [k]} \{jv \mid v \in V_\sigma(t_j)\}$, it follows that $\sum_{u \in O} \prod F_{t,u}^\Phi$ is equal to $\sum_{v \in V_\sigma(t_j), j \in [k]} \prod F_{t_j,v}^\Phi$, and similarly for O_i. We can apply the induction hypothesis for t_j to $S_{1,j}^\sigma + S_{2,j}^\sigma$, where $S_{1,j}^\sigma = \sum_{v \in V_\sigma(t_j)} \prod F_{t_j,v}^\Phi$ if $\sigma \notin \Sigma_n$, and otherwise $S_{1,j}^\sigma = 0$, and $S_{2,j}^\sigma = \sum_{v \in V_{\sigma_i}(t_j), i \in [n]} \#_\sigma(s_i) \cdot \prod F_{t_j,v}^\Phi$ if $\sigma \notin Y$, and otherwise $S_{2,j}^\sigma = 0$. Since $O_{\varepsilon,i} = \varnothing$ we get that $T_2 = 0$ and hence

$$S_1^\sigma + S_2^\sigma = T_1 + \sum_{j \in [k]} \#_\sigma(t_j\Phi).$$

Now T_1 equals 1 if $\sigma' = \sigma$, and 0 otherwise. By the definition of $\#_\sigma$ this means that the above is equal to $\#_\sigma(\sigma'(t_1\Phi, \ldots, t_k\Phi))$. This equals $\#_\sigma(t\Phi)$ by the definition of second-order tree substitution.

Case 2. $\sigma' = \sigma_i$ for some $i \in [n]$.

For every $j \in [k]$ and $v \in V(t_j)$, $\prod F_{t,jv}^\Phi = \#_{y_j}(s_i) \cdot \prod F_{t_j,v}^\Phi$. Thus, $S_1 = \sum_{j \in [k]} \#_{y_j}(s_i) \cdot S_{1,j}^\sigma$ and $S_2 = \sum_{j \in [k]} \#_{y_j}(s_i) \cdot S_{2,j}^\sigma$. By induction, $S_{1,j}^\sigma + S_{2,j}^\sigma = \#_\sigma(t_j\Phi)$. Hence $S_1 + S_2 = \sum_{j \in [k]} \#_{y_j}(s_i) \cdot \#_\sigma(t_j\Phi)$. Now $T_1 = 0$, and if $\sigma \notin Y$, then $T_2 = \#_\sigma(s_i)$, and otherwise $T_2 = 0$. We can apply Lemma 2.4 to $T_1 + T_2 + S_1 + S_2$ (with $a = \sigma$ and $S^a = T_2$) to obtain $\#_\sigma(s_i[y_j \leftarrow t_j\Phi \mid j \in [k]])$, which equals $\#_\sigma(t\Phi)$ by the definition of second-order tree substitution. □

Recall from section 2.2 that the second-order tree substitution $\Phi = [\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ is nondeleting if each s_i contains at least one occurrence of y_j for every $j \in [\mathrm{rank}_\Sigma(\sigma_i)]$, and nonerasing if no s_i is in Y. We can now use Lemma 2.6 to prove that if Φ is productive, i.e., both nondeleting and nonerasing, then its application does not decrease the size of a tree.

LEMMA 2.7. *Let Σ be a ranked alphabet and $\Phi = [\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ a second-order tree substitution over Σ. If Φ is productive, then size$(t\Phi) \geq$ size(t) for every $t \in T_\Sigma$.*

Proof. Let $\Sigma_n = \{\sigma_1, \ldots, \sigma_n\}$. Since $\mathrm{size}(t\Phi) = \sum_{\sigma \in \Sigma} \#_\sigma(t\Phi)$, we can apply Lemma 2.6 to obtain $\sum_{\sigma \in \Sigma} S_1^\sigma + \sum_{\sigma \in \Sigma} S_2^\sigma$, where S_1^σ and S_2^σ are as in Lemma 2.6.

Since Φ is nondeleting, for every $u \in V_\sigma(t)$, $\prod F_{t,u}^\Phi \geq 1$. Thus

$$\mathrm{size}(t\Phi) \geq \sum_{\sigma \in \Sigma - \Sigma_n, u \in V_\sigma(t)} 1 + \sum_{u \in V_{\sigma_i}(t), i \in [n]} \sum_{\sigma \in \Sigma - Y} \#_\sigma(s_i).$$

Since Φ is nonerasing, each s_i contains at least one symbol in $\Sigma - Y$. Hence

$$\text{size}(t\Phi) \geq \sum_{\sigma\in\Sigma-\Sigma_n, u\in V_\sigma(t)} 1 + \sum_{\sigma\in\Sigma_n, u\in V_\sigma(t)} 1 = \sum_{\sigma\in\Sigma, u\in V_\sigma(t)} 1 = \text{size}(t). \qquad \square$$

2.4. Tree languages. A *tree language* is a subset of T_Σ for some ranked alphabet Σ.

A *finite state tree automaton* is a tuple (P, Σ, h), where P is a finite set of *states*, Σ is a ranked alphabet of *input symbols* such that Σ is disjoint with P, and h is a collection of mappings such that for every $\sigma \in \Sigma^{(k)}$, h_σ is a mapping from P^k to P. The extension $\tilde{h}$ of h to a mapping from T_Σ to P is recursively defined as $\tilde{h}(\sigma(s_1, \ldots, s_k)) = h_\sigma(\tilde{h}(s_1), \ldots, \tilde{h}(s_k))$ for every $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $s_1, \ldots, s_k \in T_\Sigma$. Throughout this paper we simply write $h(s)$ to mean $\tilde{h}(s)$ for $s \in T_\Sigma$. A tree language L is *regular* (or *recognizable*) if there is a finite state tree automaton (P, Σ, h) and a subset F of P such that $L = \{s \in T_\Sigma \mid h(s) \in F\}$. For $p \in P$ the tree language $\{s \in T_\Sigma \mid h(s) = p\}$ is denoted by L_p. Note that, in particular, L_p is regular for every $p \in P$.

2.5. Tree translations. Let Σ and Δ be ranked alphabets. A (total) function $\tau : T_\Sigma \to T_\Delta$ is called a *tree translation* or simply a translation. For a tree language $L \subseteq T_\Sigma$, $\tau(L)$ denotes the set $\{t \in T_\Delta \mid t = \tau(s) \text{ for some } s \in L\}$, and for $L \subseteq T_\Delta$, $\tau^{-1}(L)$ denotes $\{s \in T_\Sigma \mid \tau(s) \in L\}$. For a class $\mathcal{T}$ of tree translations and a class $\mathcal{L}$ of tree languages, $\mathcal{T}(\mathcal{L})$ denotes the class of tree languages $\{\tau(L) \mid \tau \in \mathcal{T}, L \in \mathcal{L}\}$.

A tree translation $\tau : T_\Sigma \to T_\Delta$ is *of linear size increase* (for short, *lsi*) if there is a $c \in \mathbb{N}$ such that $\text{size}(\tau(s)) \leq c \cdot \text{size}(s)$ for all $s \in T_\Sigma$. The class of all tree translations of linear size increase is denoted *LSI*.

We will now shortly define MSO definability of a tree translation. This definition will, however, not be needed in the paper. Let k be the maximal rank of a symbol in Δ. The tree translation $\tau : T_\Sigma \to T_\Delta$ is *MSO definable* (i.e., definable in monadic second-order logic) if there is an *MSO tree transducer* which realizes τ, that is, if there exist a finite set C and MSO(Σ)-formulas $\nu_c(x)$, $\psi_{\delta,c}(x)$, and $\chi_{i,c,d}(x, y)$, with $c, d \in C$, $\delta \in \Delta$, and $1 \leq i \leq k$, such that for every $s \in T_\Sigma$, $\tau(s) \in T_\Delta$ is isomorphic to the tree t with set of nodes $\{(c, x) \in C \times V(s) \mid s \models \nu_c(x)\}$, node (c, x) has label δ if and only if $s \models \psi_{\delta,c}(x)$, and (d, y) is the ith child of (c, x) if and only if $s \models \chi_{i,c,d}(x, y)$. An MSO($\Sigma$)-formula is a formula of MSO logic that uses atomic formulas $\text{lab}_\sigma(x)$ and $\text{child}_i(x, y)$, with $\sigma \in \Sigma$ and $i \geq 1$, to express that x has label σ and y is the ith child of x, respectively. The class of all MSO definable tree translations is denoted *MSOTT*. For examples and more details, see, e.g., [5, 3]. Note that, by definition, every MSO definable tree translation τ is of linear size increase: $\text{size}(\tau(s)) \leq |C| \cdot \text{size}(s)$. Thus, $MSOTT \subseteq LSI$.

3. Macro tree transducers. In this section we recall the definition of MTTs and some basic lemmas about them. Furthermore, we consider two subclasses of MTTs which are defined by certain (static) restrictions on the rules of the transducers.

3.1. Basic definitions and results. A macro tree transducer is a syntax-directed translation device in which the translation of an input tree may depend on its subtrees, represented by input variables $x_1, x_2, \ldots$, and on its context, represented by parameters $y_1, y_2, \ldots$. A rule of an MTT is of the form $\langle q, \sigma(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to \zeta$, where q is a state of rank m and σ is an input symbol of rank k. The left-hand side will be viewed as a tree with a root, labeled $\langle q, \sigma(x_1, \ldots, x_k)\rangle$, and m children, labeled

$y_1, \ldots, y_m$, respectively. The right-hand side ζ is a tree over output symbols, the parameters $y_1, \ldots, y_m$, and symbols $\langle q', x_i \rangle$, where q' is a (ranked) state and $1 \leq i \leq k$. Thus, the MTT generalizes the top-down tree transducer in the sense that states have parameters. Note that symbols $\langle q', x_i \rangle$ can occur at any node of ζ, not just at leaves as in the case of top-down tree transducers.

We consider only *total deterministic* MTTs. For technical reasons we add the feature of regular look-ahead to them (this does not change the class of translations; cf. Theorem 4.21 of [24]). Regular look-ahead means that in a rule as above the input variables $x_1, \ldots, x_k$ range over regular tree languages. Formally, these regular tree languages are combined into one finite state tree automaton.

DEFINITION 3.1 (MTT with regular look-ahead). *A* macro tree transducer with regular look-ahead *(for short, MTT$^{\mathrm{R}}$) is a tuple $M = (Q, P, \Sigma, \Delta, q_0, R, h)$, where Q is a ranked alphabet of* states, *Σ and Δ are ranked alphabets of* input *and* output symbols, *respectively, $\Delta \cap Y = \varnothing$, $q_0 \in Q^{(0)}$ is the* initial state, *(P, Σ, h) is a finite state tree automaton, called the* look-ahead automaton of *M, and R is a finite set of* rules *of the following form: For every $q \in Q^{(m)}$, $\sigma \in \Sigma^{(k)}$, and $p_1, \ldots, p_k \in P$ with $m, k \geq 0$ there is exactly one rule of the form*

$$(*) \qquad \langle q, \sigma(x_1, \ldots, x_k) \rangle (y_1, \ldots, y_m) \to \zeta \quad \langle p_1, \ldots, p_k \rangle$$

in R, where $\zeta \in T_{\langle Q, X_k \rangle \cup \Delta}(Y_m)$.

A rule r of the form $(*)$ is called the $(q, \sigma, \langle p_1, \ldots, p_k \rangle)$-rule and its right-hand side ζ is denoted by $\mathrm{rhs}(r)$ or by $\mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k \rangle)$; it is also called a q-rule, a σ-rule, or a (q, σ)-rule. A *top-down tree transducer with regular look-ahead* (for short, T$^{\mathrm{R}}$) is an MTT$^{\mathrm{R}}$ all states of which are of rank zero. If the look-ahead automaton is trivial, i.e., $P = \{p\}$ and $h_\sigma(p, \ldots, p) = p$ for all $\sigma \in \Sigma$, then M is called an *MTT*, and if M is a T$^{\mathrm{R}}$, then M is called a *top-down tree transducer*. In such cases we omit the look-ahead automaton and simply denote M by $(Q, \Sigma, \Delta, q_0, R)$; we also omit the look-ahead part $\langle p_1, \ldots, p_k \rangle$ in every rule $(*)$.

We now define the derivation relation induced by an MTT$^{\mathrm{R}}$ M. Recall from section 2.2 that in a second-order tree substitution $\langle q', x_i \rangle \leftarrow \langle q', s_i \rangle$ is shorthand for $\langle q', x_i \rangle \leftarrow \langle q', s_i \rangle (y_1, \ldots, y_n)$, where n is the rank of q'.

DEFINITION 3.2 (derivation relation). *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R. The* derivation relation induced by *M, denoted by $\Rightarrow_M$, is the binary relation on $T_{\langle Q, T_\Sigma \rangle \cup \Delta}(Y)$ such that, for every $\xi_1, \xi_2 \in T_{\langle Q, T_\Sigma \rangle \cup \Delta}(Y)$, $\xi_1 \Rightarrow_M \xi_2$ if and only if there exist $u \in V(\xi_1)$, $\sigma \in \Sigma^{(k)}$, $s_1, \ldots, s_k \in T_\Sigma$, $q \in Q^{(m)}$, and $t_1, \ldots, t_m \in T_{\langle Q, T_\Sigma \rangle \cup \Delta}(Y)$ such that $\xi_1/u = \langle q, \sigma(s_1, \ldots, s_k) \rangle (t_1, \ldots, t_m)$ and $\xi_2 = \xi_1[u \leftarrow \zeta]$, where ζ equals*

$$\mathrm{rhs}_M(q, \sigma, \langle h(s_1), \ldots, h(s_k) \rangle) [\![\langle q', x_i \rangle \leftarrow \langle q', s_i \rangle \mid \langle q', x_i \rangle \in \langle Q, X_k \rangle]\!] [y_j \leftarrow t_j \mid j \in [m]].$$

Since the derivation relation $\Rightarrow_M$ induced by M is confluent and terminating (cf., e.g., Chapter 4 of [28]) there is, for every tree $\xi \in T_{\langle Q, T_\Sigma \rangle \cup \Delta}(Y_m)$, a unique tree $t \in T_\Delta(Y_m)$ such that $\xi \Rightarrow_M^* t$ (in fact, t is the normal form of ξ with respect to $\Rightarrow_M$).

DEFINITION 3.3 (translation). *For every $q \in Q^{(m)}$ and $s \in T_\Sigma$ the* q-translation of *s, denoted by $M_q(s)$, is the unique tree $t \in T_\Delta(Y_m)$ such that $\langle q, s \rangle (y_1, \ldots, y_m) \Rightarrow_M^* t$. Thus, for $q \in Q^{(m)}$, M_q is a (total) function from T_Σ to $T_\Delta(Y_m)$. The* translation realized by *M, denoted by τ_M, is the (total) function $M_{q_0} : T_\Sigma \to T_\Delta$.*

Thus, $\tau_M = M_{q_0}$ and for $s \in T_\Sigma$, $\tau_M(s) = M_{q_0}(s)$ is the unique tree $t \in T_\Delta$ such that $\langle q_0, s \rangle \Rightarrow_M^* t$. If $\tau_M(s) = t$, then s is an *input tree* (of M) and t is the corresponding *output tree* (of M).

An $\mathrm{MTT}^{\mathrm{R}}$ is *of linear size increase* (for short, lsi) if τ_M is lsi (cf. section 2.5). Two $\mathrm{MTT}^{\mathrm{R}}$s M and M' are *equivalent* if $\tau_M = \tau_{M'}$. The class of all translations which can be realized by MTTs and $\mathrm{MTT}^{\mathrm{R}}$s is denoted by MTT and MTT^R, respectively. The class of all translations which can be realized by T^{R}s is denoted by T^R.

LEMMA 3.4 (Theorem 4.21 of [24]). *$MTT^R = MTT$ (effectively).*

The q-translations $M_q(s)$ of trees $s \in T_\Sigma$ can be characterized inductively as follows, using second-order tree substitution.

LEMMA 3.5 (Lemma 4.8 of [26]). *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R. For every $q \in Q$, $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $s_1, \ldots, s_k \in T_\Sigma$,*

$$M_q(\sigma(s_1, \ldots, s_k)) = \mathrm{rhs}_M(q, \sigma, \langle h(s_1), \ldots, h(s_k)\rangle)[\![\langle q', x_i\rangle \leftarrow M_{q'}(s_i) \mid \langle q', x_i\rangle \in \langle Q, X_k\rangle]\!].$$

As mentioned in the introduction, macro tree translations can be of double exponential size increase. This is shown in the following example.

Example 3.6. Let $M = (Q, \Sigma, \Delta, q_0, R)$ be the MTT with $Q = \{q_0^{(0)}, q^{(1)}\}$, $\Sigma = \{\sigma^{(1)}, \alpha^{(0)}\}$, $\Delta = \{\delta^{(2)}, \beta^{(0)}\}$, and R consisting of the following rules:

$$\begin{array}{lcl} \langle q_0, \sigma(x_1)\rangle & \to & \langle q, x_1\rangle(\beta), \\ \langle q_0, \alpha\rangle & \to & \beta, \\ \langle q, \sigma(x_1)\rangle(y_1) & \to & \langle q, x_1\rangle(\langle q, x_1\rangle(y_1)), \\ \langle q, \alpha\rangle(y_1) & \to & \delta(y_1, y_1). \end{array}$$

The MTT M translates α into β, and for $n \geq 0$ it translates the input tree $s_n = \sigma(\sigma^n(\alpha))$ into a full binary tree of height $2^n + 1$ (with 2^{2^n} leaves). Figure 3.1 shows a derivation of M: First $\langle q_0, s_n\rangle \Rightarrow_M \langle q, \sigma^n(\alpha)\rangle(\beta)$. Then, due to the copying of states of the (q, σ)-rule, $\langle q, \sigma^n(\alpha)\rangle(\beta)$ is translated into the monadic tree $\langle q, \alpha\rangle(\langle q, \alpha\rangle(\cdots\langle q, \alpha\rangle(\beta)\cdots))$ containing 2^n occurrences of $\langle q, \alpha\rangle$. At last, due to the copying of parameters of the (q, α)-rule, this monadic tree is translated into a full binary tree of height $2^n + 1$. Thus, the input tree s_n of size $n + 2$ is translated into a tree of size $2^{2^n+1} - 1$, and hence the translation realized by M is of double exponential size increase. Note that the q-translation $M_q(\sigma^n(\alpha)) \in T_\Delta(\{y_1\})$ is the full binary tree of height $2^n + 1$ in which each leaf is labeled y_1, denoted t_n in what follows. In fact, a derivation $\langle q, \sigma^n(\alpha)\rangle(y_1) \Rightarrow_M \langle q, \sigma^{n-1}(\alpha)\rangle(\langle q, \sigma^{n-1}(\alpha)\rangle(y_1)) \Rightarrow_M^* t_n$ can be obtained from Figure 3.1 by removing the first derivation step and changing every β into y_1. Another way of showing that $M_q(\sigma^n(\alpha)) = t_n$ is by induction on n, using Lemma 3.5. For $n = 0$, $M_q(\alpha) = \mathrm{rhs}_M(q, \alpha) = \delta(y_1, y_1) = t_0$. The induction step is proved as follows: $M_q(\sigma(\sigma^n(\alpha))) = \mathrm{rhs}_M(q, \sigma)[\![\langle q, x_1\rangle \leftarrow M_q(\sigma^n(\alpha))]\!] = \langle q, x_1\rangle(\langle q, x_1\rangle(y_1))[\![\langle q, x_1\rangle \leftarrow t_n]\!] = t_n[y_1 \leftarrow t_n] = t_{n+1}$. This ends the example.

The following two results are often used in this paper.

LEMMA 3.7 (Lemma 7.4(1) of [24]). *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R. For every $q \in Q^{(m)}$, $m \geq 0$, and regular tree language $L \subseteq T_\Delta(Y_m)$, $M_q^{-1}(L)$ is regular and can be defined effectively.*

Proof. In Lemma 7.4(1) of [24] the result is stated for the case $m = 0$. The general case can be reduced to this case as follows: For every $r \in Q$ let $\overline{r}$ be a symbol not in Σ. Define the $\mathrm{MTT}^{\mathrm{R}}$ $\overline{M} = (Q, P, \Sigma \cup \{\overline{r}^{(1)} \mid r \in Q\}, \Delta \cup \{\overline{y}_j^{(0)} \mid j \in [\overline{m}]\}, q_0, R \cup \overline{R}, h \cup \overline{h})$, where $\overline{m}$ is the maximal rank of a state of M. For every $r \in Q^{(n)}$, $n \geq 0$, and $p \in P$ let $\overline{h}_{\overline{r}}(p) = p$, and let the rule

$$\langle q_0, \overline{r}(x_1)\rangle \to \langle r, x_1\rangle(\overline{y}_1, \ldots, \overline{y}_n) \quad \langle p\rangle$$

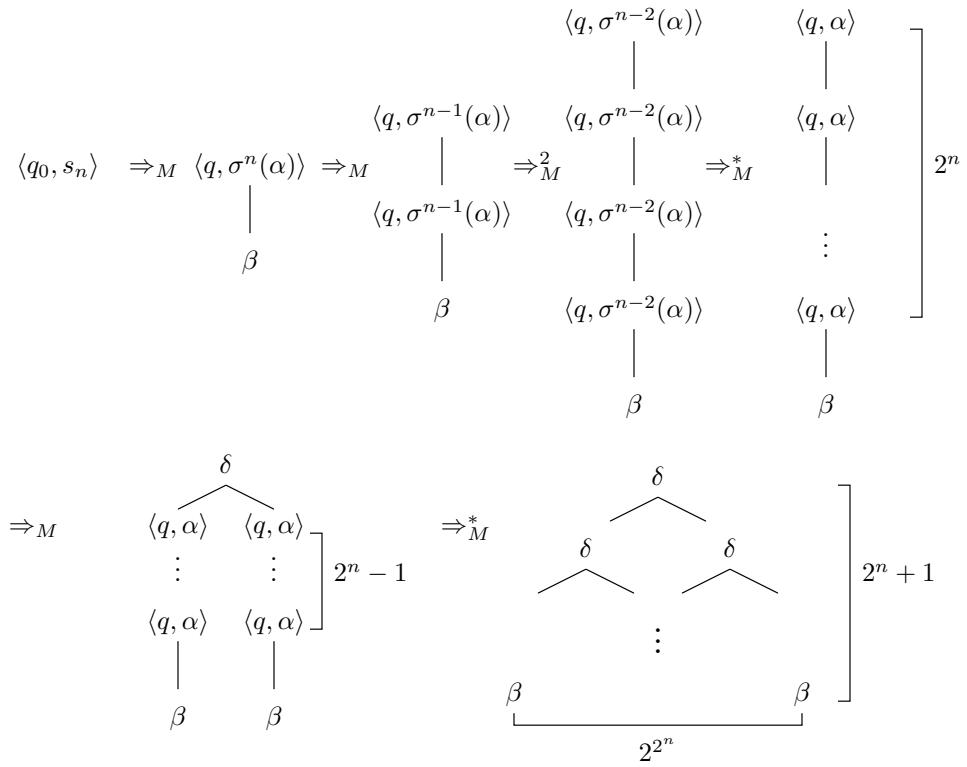


FIG. 3.1. *Derivation by* $\Rightarrow_M$.

be in $\overline{R}$. Clearly, $\tau_{\overline{M}}(\overline{r}(s)) = M_r(s)[y_j \leftarrow \overline{y}_j \mid j \in [n]]$ for every $s \in T_\Sigma$. Let $\overline{L} = \{t[y_j \leftarrow \overline{y}_j \mid j \in [m]] \mid t \in L\}$. By Lemma 7.4(1) of [24], $\tau_{\overline{M}}^{-1}(\overline{L})$ is (effectively) regular. Then also $\tau_{\overline{M}}^{-1}(\overline{L}) \cap \overline{q}(T_\Sigma) = \overline{q}(M_q^{-1}(L))$ is (effectively) regular (because regular tree languages are effectively closed under intersection; cf., e.g., Theorem II.4.2 of [31]). Since there is a linear top-down tree transducer that translates each tree $\overline{q}(t)$ into the tree t, and regular tree languages are (effectively) closed under linear top-down tree translations (see, e.g., Corollary IV.6.6 of [31]), we obtain that $M_q^{-1}(L)$ is (effectively) regular. □

The next lemma follows from Theorem 4.5 of [12] and Theorem 7.3 of [24] (and from the obvious fact that every regular tree language is the range of a nondeterministic top-down tree transducer; cf., e.g., Proposition 20.1(ii) of [32]). Note that we have not defined nondeterministic MTT$^{\mathrm{R}}$s and that we need to apply Lemma 3.8 only once to a nondeterministic (top-down) tree transducer (in Lemma 5.7).

LEMMA 3.8 (Theorem 4.5 of [12]). *For a regular tree language L and a finite number of (possibly nondeterministic) MTT^{R}s $M_1, \ldots, M_n$ it is decidable whether or not $\tau_{M_n}(\tau_{M_{n-1}}(\cdots \tau_{M_1}(L) \cdots))$ is finite. Moreover, if it is finite, it can be constructed.*

3.2. Subclasses defined by restrictions on the parameters. We now define two restrictions on the occurrences of parameters in the right-hand sides of the rules of an MTT$^{\mathrm{R}}$ M and then show that these restrictions carry over to the q-translations $M_q(s)$ of M.

DEFINITION 3.9 (nondeleting, nonerasing, productive). *Let* $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ *be an* MTT^R. *If for every* $q \in Q^{(m)}$, $m \geq 1$, $\sigma \in \Sigma^{(k)}$, $k \geq 0$, $p_1, \ldots, p_k \in P$, *and* $j \in [m]$,

- y_j *occurs at least once in* $\mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k \rangle)$, *then* M *is* nondeleting;
- $\mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k \rangle) \notin Y$, *then* M *is* nonerasing.

If M *is both nondeleting and nonerasing, then it is* productive.

LEMMA 3.10 (Lemma 7.11 of [19]). *For every* MTT^R M *there is a productive* MTT^R M' *equivalent to* M.

The following lemma shows that the restrictions nondeleting and nonerasing carry over from the right-hand sides of an $\mathrm{MTT^R}$ to the q-translations of M. In Lemma 6.7 of [19] a similar result is proved: if in the right-hand side of every q-rule each parameter y_j of q occurs exactly once, then y_j occurs exactly once in $M_q(s)$.

LEMMA 3.11. *Let* $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ *be an* MTT^R. *For every* $q \in Q^{(m)}$, $m \geq 0$, $j \in [m]$, *and* $s \in T_\Sigma$,

(1) *if* M *is nondeleting, then* $\#_{y_j}(M_q(s)) \geq 1$; *and*

(2) *if* M *is nonerasing, then* $M_q(s) \notin Y$.

Proof. The proof is by induction on the structure of s. Let $s = \sigma(s_1, \ldots, s_k)$ with $k \geq 0$ and $s_1, \ldots, s_k \in T_\Sigma$. Denote by t the tree $\mathrm{rhs}_M(q, \sigma, \langle h(s_1), \ldots, h(s_k) \rangle)$. By Lemma 3.5, $M_q(s) = t\Phi$ with $\Phi = [\![\langle q', x_i \rangle \leftarrow M_{q'}(s_i) \mid \langle q', x_i \rangle \in \langle Q, X_k \rangle]\!]$.

(1) By induction $\#_{y_\nu}(M_{q'}(s_i)) \geq 1$ for all $\langle q', x_i \rangle \in \langle Q, X_k \rangle^{(n)}$ and $\nu \in [n]$; i.e., the substitution Φ is nondeleting. Since M is nondeleting, $\#_{y_j}(t) \geq 1$ and thus, by Lemma 2.1, $\#_{y_j}(t\Phi) \geq 1$.

(2) By induction $M_{q'}(s_i) \notin Y$ for all $\langle q', x_i \rangle \in \langle Q, X_k \rangle$; i.e., the substitution Φ is nonerasing. Since M is nonerasing, $t \notin Y$ and thus, by Lemma 2.2, $t\Phi \notin Y$. □

4. Finite copying restrictions. In this section we define various restrictions on the copying that is performed by an $\mathrm{MTT^R}$. First, in section 4.1, copying restrictions for the input variables and for the parameters are defined. Both together form the "finite copying" restriction which was introduced in [19]; there it was shown (in Theorem 7.1) that the translations realized by finite copying $\mathrm{MTT^R}$s are precisely the MSO definable tree translations (cf. section 2.5). Since, by their definition, the MSO definable tree translations are lsi, this means that finite copying $\mathrm{MTT^R}$s are lsi. To keep this paper self-contained, we give, in section 4.3, a direct proof of this fact, which is based on the notion of "finite contribution." Intuitively, an $\mathrm{MTT^R}$ is of finite contribution if there is a bound on the number of output nodes contributed by a single node u of the input tree. In the terminology of [50], the node u is called the "origin" of the nodes of the output tree that it contributes; so, finite contribution means that there is a bound on the number of nodes that have the same origin. In [50] it is shown that for a primitive recursive scheme, which is an MTT, every node of an output tree has exactly one origin.

We also define, in section 4.2, a restriction on the copying that occurs on one path of the output tree, i.e., a restriction on the amount of nesting of states that occurs during the derivation of an $\mathrm{MTT^R}$. This notion will play an essential role in section 6, where it is proved that if the translation of an $\mathrm{MTT^R}$ is lsi, then it can also be realized by a finite copying $\mathrm{MTT^R}$ (and hence is MSO definable).

4.1. Finite copying in the input and in the parameters. Here we recall the definition of finite copying $\mathrm{MTT^R}$s from [19] and show that for an $\mathrm{MTT^R}$ it is decidable whether or not it is finite copying. The finite copying restriction was introduced in [1] for generalized syntax-directed translation schemes. For top-down tree transducers it was investigated in [22]. A top-down tree transducer is finite copying if every subtree

of the input tree is translated by boundedly many occurrences of states, i.e., the length of the state sequence is bounded, where the state sequence at a subtree s/u consists of the states that translate s/u. For an MTT this restriction is called finite copying in the input (fci), and we additionally have a restriction for the parameters, called finite copying in the parameters (fcp). The fcp restriction requires that, for every state q and input tree s, the number of parameters that occur in the q-translation $M_q(s)$ of s is bounded.

In order to define the state sequence of a tree s at the node u of s, we first extend an $\mathrm{MTT}^{\mathrm{R}}$ in such a way that the output tree t, for the input tree $s[u \leftarrow p]$, contains the states which process the subtree s/u (assuming that $p = h(s/u)$). More precisely, t contains $\langle\langle q, p\rangle\rangle$ if the state q translates s/u. Analogous to the definition of $\langle \Sigma, A\rangle$ let, for a ranked set Σ and a set A, $\langle\langle \Sigma, A\rangle\rangle$ be the ranked set of all symbols $\langle\langle \sigma, a\rangle\rangle$ of rank m for $\sigma \in \Sigma^{(m)}$ and $a \in A$.

Definition 4.1 (Definition 3.5 of [19]: extension). *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R. The* extension of M, *denoted by $\hat{M}$, is the MTT^R $(Q, P, \hat{\Sigma}, \hat{\Delta}, q_0, \hat{R}, \hat{h})$, where $\hat{\Sigma} = \Sigma \cup \{p^{(0)} \mid p \in P\}$, $\hat{\Delta} = \Delta \cup \langle\langle Q, P\rangle\rangle$, $\hat{R} = R \cup \{\langle q, p\rangle(y_1, \ldots, y_m) \to \langle\langle q, p\rangle\rangle(y_1, \ldots, y_m) \mid \langle q, p\rangle \in \langle Q, P\rangle^{(m)}\}$, $\hat{h}_p() = p$ for $p \in P$, and $\hat{h}_\sigma(p_1, \ldots, p_k) = h_\sigma(p_1, \ldots, p_k)$ for $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $p_1, \ldots, p_k \in P$.*

Note that if M is nondeleting or nonerasing, then so is $\hat{M}$. Before state sequences and the fci and fcp properties are defined, we present two useful lemmas about the q-translations of $\hat{M}$. The first lemma shows that the q-translation of an input tree s can be obtained by replacing in the q-translation of the "context" of a node u of s, $\hat{M}_q(s[u \leftarrow p])$, each occurrence of $\langle\langle q', p\rangle\rangle$ by the q'-translation $M_{q'}(s/u)$ of the subtree of s at u. In fact, the lemma is stated in the more general case that s/u may contain occurrences of symbols in P. The lemma can be seen as a generalization of Lemma 3.5 from the application of a rule at the root of s, to the translation of the context of an arbitrary node u.

Lemma 4.2 (Lemma 3.6 of [19]). *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R and $\hat{M} = (Q, P, \hat{\Sigma}, \hat{\Delta}, q_0, \hat{R}, \hat{h})$ its extension. Let $q \in Q$, $s \in T_{\hat{\Sigma}}$, $u \in V(s)$, and $p = \hat{h}(s/u)$ such that $s[u \leftarrow p]$ contains exactly one occurrence of an element of P. Then*

$$\hat{M}_q(s) = \hat{M}_q(s[u \leftarrow p])[\![\langle\langle q', p\rangle\rangle \leftarrow \hat{M}_{q'}(s/u) \mid q' \in Q]\!].$$

The next lemma is obtained by application of Lemma 3.5 to the $\hat{M}_{q'}(s/u)$ in the substitution of Lemma 4.2. It shows how to express the translation of the context of a child node in terms of the translation of the context of its parent and the translations of the subtrees of its siblings.

Lemma 4.3. *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R. Let $q \in Q$, $s \in T_\Sigma$, and $u \in V(s)$. If $s[u] = \sigma \in \Sigma^{(k)}$, $i \in [k]$, $p_i \in P$, $p_j = h(s/uj)$ for every $j \in [k] - \{i\}$, and $p = h_\sigma(p_1, \ldots, p_k)$, then*

$$\hat{M}_q(s[ui \leftarrow p_i]) = \hat{M}_q(s[u \leftarrow p])[\![rhs]\!][\![..]\!][\![i]\!],$$

where

$$\begin{aligned} [\![\mathrm{rhs}]\!] &= [\![\langle\langle q', p\rangle\rangle \leftarrow \mathrm{rhs}_M(q', \sigma, \langle p_1, \ldots, p_k\rangle) \mid q' \in Q]\!], \\ [\![..]\!] &= [\![\langle r, x_j\rangle \leftarrow M_r(s/uj) \mid r \in Q, j \in [k] - \{i\}]\!], \text{ and} \\ [\![i]\!] &= [\![\langle r, x_i\rangle \leftarrow \langle\langle r, p_i\rangle\rangle \mid r \in Q]\!]. \end{aligned}$$

Proof. Let $s' = s[ui \leftarrow p_i]$. Since $p = \hat{h}(s'/u)$ and $s'[u \leftarrow p]$ contains exactly one occurrence of an element of P, we can apply Lemma 4.2 to get $\hat{M}_q(s') = \hat{M}_q(s[u \leftarrow$

$p])[\![\langle\!\langle q', p\rangle\!\rangle \leftarrow \hat{M}_{q'}(s'/u) \mid q' \in Q]\!]$. Now $s'/u = \sigma(s_1, \ldots, s_k)$ with $s_i = p_i$ and $s_j = s/uj$ for every $j \in [k] - \{i\}$. By application of Lemma 3.5 to $\hat{M}_{q'}(s'/u)$ the above equals $\hat{M}_q(s[u \leftarrow p])[\![\langle\!\langle q', p\rangle\!\rangle \leftarrow \mathrm{rhs}_M(q', \sigma, \langle p_1, \ldots, p_k\rangle)[\![\ldots]\!] \mid q' \in Q]\!]$, where $[\![\ldots]\!]$ denotes $[\![\langle r, x_j\rangle \leftarrow \hat{M}_r(s_j) \mid r \in Q, j \in [k]]\!]$. We now use the associativity of second-order tree substitution; cf. section 2.2. Since $\hat{M}_q(s[u \leftarrow p])$ does not contain elements of $\langle Q, X_k\rangle$ we can move $[\![\ldots]\!]$ out of the substitution to get $\hat{M}_q(s[u \leftarrow p])[\![\mathrm{rhs}]\!][\![\ldots]\!]$. For every $j \in [k] - \{i\}$, $\hat{M}_r(s_j) = M_r(s_j)$ does not contain elements of $\langle Q, \{x_i\}\rangle$; moreover, $\hat{M}_r(s_i) = \langle\!\langle r, p_i\rangle\!\rangle$. Thus we can write $[\![\ldots]\!]$ as $[\![..]\!][\![i]\!]$. ∎

We now turn to the definition of state sequence and the finite copying properties. Recall that the preorder of the nodes of a tree is denoted by $<$.

Definition 4.4 (Definition 3.7 of [19]: state sequence). *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R, $s \in T_\Sigma$, and $u \in V(s)$. Let $p = h(s/u)$ and $\xi = \hat{M}_{q_0}(s[u \leftarrow p]) \in T_{\langle\!\langle Q,\{p\}\rangle\!\rangle \cup \Delta}$, and let $\{v \in V(\xi) \mid \xi[v] \in \langle\!\langle Q, \{p\}\rangle\!\rangle\} = \{v_1, \ldots, v_n\}$ with $v_1 < \cdots < v_n$. The* state sequence of s at u, *denoted by $\mathrm{sts}_M(s, u)$, is the sequence of states $q_1 \cdots q_n$ such that $\xi[v_i] = \langle\!\langle q_i, p\rangle\!\rangle$ for every $i \in [n]$.*

Observe that $|\mathrm{sts}_M(s, u)| = \#_{\langle\!\langle Q,\{p\}\rangle\!\rangle}(\hat{M}_{q_0}(s[u \leftarrow p]))$, where $p = h(s/u)$.

Definition 4.5 (Definition 6.1 of [19]: fci). *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R. Then M is* fci *if there is an $N \in \mathbb{N}$ such that for every $s \in T_\Sigma$ and $u \in V(s)$: $|\mathrm{sts}_M(s, u)| \le N$. The number N is an* input copying bound for M.

Definition 4.6 (Definition 6.2 of [19]: fcp). *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R. Then M is* fcp *if there is an $N \in \mathbb{N}$ such that for every $q \in Q^{(m)}$, $s \in T_\Sigma$, and $j \in [m]$, $\#_{y_j}(M_q(s)) \le N$. The number N is a* parameter copying bound for M.

Note that the MTT M of Example 3.6 is neither fci nor fcp. There is exponential state copying: the state sequence $\mathrm{sts}_M(s_n, 11^n)$ of $s_n = \sigma(\sigma^n(\beta))$ at 11^n equals q^{2^n}, and there is double exponential parameter copying: $\#_{y_1}(M_q(\sigma^n(\beta))) = 2^{2^n}$.

The following lemma shows that if M is fcp, i.e., if the number of occurrences of y_j in $M_q(s)$ is bounded by some N, for all states q and parameters y_j of q, then also for the q-translations of $\hat{M}$ of input trees $s[u \leftarrow p]$, the number of occurrences of y_j is bounded by N. However, we must assume that M is nondeleting.

Lemma 4.7. *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be a nondeleting fcp MTT^R and let N be a parameter copying bound for M. For every $q \in Q^{(m)}$, $j \in [m]$, $s \in T_\Sigma$, and $u \in V(s)$, $\#_{y_j}(\hat{M}_q(s[u \leftarrow h(s/u)])) \le N$.*

Proof. Let $p = h(s/u)$. By Lemma 4.2, $M_q(s) = \xi[\![\ldots]\!]$ with $\xi = \hat{M}_q(s[u \leftarrow p])$ and $[\![\ldots]\!] = [\![\langle\!\langle q', p\rangle\!\rangle \leftarrow M_{q'}(s/u) \mid q' \in Q]\!]$. By Lemma 2.6, $\#_{y_j}(\xi[\![\ldots]\!]) = \sum_{v \in V_{y_j}(\xi)} \prod F_{\xi,v}^{[\![\ldots]\!]}$. Let $V_{y_j}(\xi) = \{v_1, \ldots, v_n\}$. Then the above sum equals

$$\prod F_{\xi,v_1}^{[\![\ldots]\!]} + \cdots + \prod F_{\xi,v_n}^{[\![\ldots]\!]} = \#_{y_j}(M_q(s)) \le N,$$

which implies that $n = \#_{y_j}(\xi) \le N$ because $\prod F_{\xi,v_i}^{[\![\ldots]\!]} \ge 1$ for every $i \in [n]$, by the fact that M is nondeleting, and hence, by Lemma 3.11(1), $\#_{y_k}(M_{q'}(s/u)) \ge 1$ for every $q' \in Q^{(m')}$ and $k \in [m']$. ∎

Finally, the combination of fci and fcp yields the finite copying property.

Definition 4.8 (finite copying). *An MTT^R is* finite copying *if it is both fci and fcp.*

We use the subscripts "fci," "fcp," or "fc" for classes of translations to denote that the corresponding MTT$^{\mathrm{R}}$s are fci, fcp, or finite copying, respectively. Thus $MTT^R_{\mathrm{fc}} = MTT^R_{\mathrm{fci,fcp}}$. The main result of [19] is that the translations of finite copying MTT$^{\mathrm{R}}$s are precisely the MSO definable tree translations (see section 2.5).

LEMMA 4.9 (Theorem 7.1 of [19]). *$MSOTT = MTT^R_{fc}$ (effectively).*

The main results of this paper are the following: (i) the translations of finite copying $\mathrm{MTT^R}$s are precisely the translations of $\mathrm{MTT^R}$s that are of linear size increase (i.e., $MTT^R \cap LSI = MTT^R_{\mathrm{fc}}$), and (ii) it is decidable for an $\mathrm{MTT^R}$ M whether or not there exists an equivalent finite copying $\mathrm{MTT^R}$ (i.e., whether $\tau_M \in MTT^R_{\mathrm{fc}}$). We now show that it is decidable for an $\mathrm{MTT^R}$ M whether or not M is finite copying. The proof is based on Lemma 3.8.

LEMMA 4.10. *It is decidable for an MTT^R M*

(i) *whether or not M is fci, and*

(ii) *whether or not M is fcp,*

and if so, a copying bound can be obtained effectively.

Proof. Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$.

(i) To decide fci we define the MTT N that, when applied to $\tau_{\hat{M}}(s[u \leftarrow h(s/u)])$, generates the state sequence $\mathrm{sts}_M(s,u)$ as a monadic tree. Let $N = (Q', \Delta \cup \langle\langle Q, P\rangle\rangle, \Gamma, r_0, R')$ with $Q' = \{r_0^{(0)}, r^{(1)}\}$ and $\Gamma = \{q^{(1)} \mid q \in Q\} \cup \{e^{(0)}\}$. For every $k \geq 0$, $\langle\langle q,p\rangle\rangle \in \langle\langle Q,P\rangle\rangle^{(k)}$, and $\delta \in \Delta^{(k)}$ let the following rules be in R':

$$\begin{array}{lcl}
\langle r_0, \langle\langle q,p\rangle\rangle(x_1,\dots,x_k)\rangle & \to & q(\langle r,x_1\rangle(\langle r,x_2\rangle(\cdots\langle r,x_k\rangle(e)\cdots))),\\
\langle r_0, \delta(x_1,\dots,x_k)\rangle & \to & \langle r,x_1\rangle(\langle r,x_2\rangle(\cdots\langle r,x_k\rangle(e)\cdots)),\\
\langle r, \langle\langle q,p\rangle\rangle(x_1,\dots,x_k)\rangle(y_1) & \to & q(\langle r,x_1\rangle(\langle r,x_2\rangle(\cdots\langle r,x_k\rangle(y_1)\cdots))),\\
\langle r, \delta(x_1,\dots,x_k)\rangle(y_1) & \to & \langle r,x_1\rangle(\langle r,x_2\rangle(\cdots\langle r,x_k\rangle(y_1)\cdots)).
\end{array}$$

Then, for every $s \in T_\Sigma$ and $u \in V(s)$, $\mathrm{lpath}(\tau_N(\tau_{\hat{M}}(s[u \leftarrow h(s/u)])), v) = \mathrm{sts}_M(s,u)e$, where v is the unique leaf of $\tau_N(\tau_{\hat{M}}(s[u \leftarrow h(s/u)]))$.

Let L be the tree language $\{s[u \leftarrow h(s/u)] \mid s \in T_\Sigma, u \in V(s)\}$. Then M is fci if and only if $K = \tau_N(\tau_{\hat{M}}(L))$ is finite. Note that $L = \{s \in T_\Sigma(P') \mid \#_{P'}(s) = 1\}$, where $P' = \{p \in P \mid L_p \neq \varnothing\}$; hence L is (effectively) regular. Thus, finiteness of K can be decided by Lemma 3.8; in case of finiteness, K can be constructed and an input copying bound for M is $\max\{\mathrm{size}(t) \mid t \in K\} - 1$.

(ii) Let $\overline{M}$ be the $\mathrm{MTT^R}$ defined in the proof of Lemma 3.7 and let $\overline{\Delta} = \Delta \cup \{\overline{y}_j^{(0)} \mid j \in [\overline{m}]\}$ be its output alphabet, where $\overline{m}$ is the maximal rank of a state of M. Let $N = (\{r_0^{(0)}, r^{(1)}\}, \overline{\Delta}, \Gamma, r_0, R_N)$ be the MTT with $\Gamma = \{\overline{y}_j^{(1)} \mid j \in [\overline{m}]\} \cup \{e^{(0)}\}$. For $\delta \in \Delta^{(k)}$ with $k \geq 0$ the (r_0,δ)- and (r,δ)-rules are defined as for N in (i). For $j \in [\overline{m}]$ let the rules $\langle r_0, \overline{y}_j\rangle \to \overline{y}_j(e)$ and $\langle r, \overline{y}_j\rangle(y_1) \to \overline{y}_j(y_1)$ be in R_N.

Clearly, for every $q \in Q$ and $s \in T_\Sigma$, $\mathrm{size}(\tau_N(\tau_{\overline{M}}(\overline{q}(s)))) = 1 + \#_Y(M_q(s))$. Now, for the regular tree language $L = \{\overline{q}(s) \mid q \in Q, s \in T_\Sigma\}$, M is fcp if and only if $K = \tau_N(\tau_{\overline{M}}(L))$ is finite. As in (i), this can be decided by Lemma 3.8; in case of finiteness, K can be constructed and a parameter copying bound for M is $\max\{\mathrm{size}(t) \mid t \in K\} - 1$. □

In fact, the effectiveness of Lemma 4.9 was not completely proved in [19], but with Lemma 4.10 it can be shown as follows: Given an $\mathrm{MTT^R_{fc}}$ M we can use Lemma 4.10 to obtain a parameter copying bound N for M. Then, given M and N we can, by the proof of Lemma 6.3 of [19], construct an $\mathrm{MTT^R_{fci,surp}}$ M' equivalent to M (where "surp" means "single-use restricted in the parameters"). Now, again by Lemma 4.10 we can determine an input copying bound N for M'. Then, given M' and N we can, by the proof of Lemma 6.10 of [19], construct a single-use restricted $\mathrm{MTT^R}$ M'' equivalent to M'. Now by the proofs of Lemmas 5.9, 5.12, and 4.1 of [19], a single-use restricted attributed tree transducer with look-ahead (for short, $\mathrm{ATT^R}$) A equivalent to M'' can be constructed. Given A, the proof of Lemma 7 of [3] shows

how to construct an equivalent MSO tree transducer. This proves the effectiveness going from MTT^R_{fc} to $MSOTT$. For the other direction, that is, starting with an MSO tree transducer M, we can proceed as follows: The proof of Theorem 14 of [3] gives a construction of an equivalent single-use restricted $\mathrm{ATT}^{\mathrm{R}}$ A. The proofs of Lemmas 4.2 and 5.11 of [19] show how to construct an equivalent single-use restricted $\mathrm{MTT}^{\mathrm{R}}$ M'. By the proof of Theorem 6.12 of [19], M' is finite copying.

4.2. Finite nested copying in the input. Consider the translation $\xi = \hat{M}_{q_0}(s[u \leftarrow p])$ of the context of a node u of the input tree s, where $p = h(s/u)$. The symbols of $\langle\langle Q, \{p\}\rangle\rangle$ can occur nested in ξ; i.e., they can occur on a common label path $\mathrm{lpath}(\xi, v)$ to some node v of ξ. Assuming that M is nondeleting, this means that a lot of copies of v will be generated; namely, $\prod F_{\xi,v}^{[\![\ldots]\!]}$ copies, where $[\![\ldots]\!]$ replaces $\langle\langle q, p\rangle\rangle$ by $M_q(s/u)$. Thus, a way to bound the copying carried out by M is to bound by some $B \in \mathbb{N}$ the number of elements of $\langle\langle Q, \{p\}\rangle\rangle$ that occur on a label path in ξ, i.e., to bound the nesting of states. This implies that the number of elements in the family $F_{\xi,v}^{[\![\ldots]\!]}$ is bounded by B. We call this property *finite nested copying in the input* (for short, fnest). Clearly, it is a much weaker restriction than the fci restriction. However, if an $\mathrm{MTT}^{\mathrm{R}}$ is fnest and fcp, then $\prod F_{\xi,v}^{[\![\ldots]\!]}$ is bounded by N^B if N is a parameter copying bound for M.

Definition 4.11 (fnest). *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R. Then M is* fnest *if there is a $B \in \mathbb{N}$ such that for every $s \in T_\Sigma$, $u \in V(s)$, $p = h(s/u)$, and label path π in $\hat{M}_{q_0}(s[u \leftarrow p])$, $\#_{\langle\langle Q,\{p\}\rangle\rangle}(\pi) \le B$. The number B is a* nesting bound for M.

We use the subscript "fnest" for classes of translations of $\mathrm{MTT}^{\mathrm{R}}$s to denote that the corresponding transducers are fnest. The next lemma shows that the nesting bound B also holds for trees $\hat{M}_q(s[u \leftarrow p])$ with $s \in L_{p'}$, provided that $\langle\langle q, p'\rangle\rangle$ is reachable in the following sense.

Definition 4.12 (reachable). *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R, $q \in Q$, and $p \in P$. Then $\langle\langle q, p\rangle\rangle$ is* reachable *if there are $s \in T_\Sigma$ and $u \in V(s)$ such that $\langle\langle q, p\rangle\rangle$ occurs in $\hat{M}_{q_0}(s[u \leftarrow p])$.*

Note that reachability does not require that $h(s/u) = p$; however, for $L_p \neq \varnothing$ this can always be assumed (simply take $s' = s[u \leftarrow t]$ for some $t \in L_p$ if $h(s/u) \neq p$). Note that in that case, q occurs in the state sequence of s at u.

Lemma 4.13. *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be a nondeleting fnest MTT^R and let B be a nesting bound for M. If $\langle\langle q, p\rangle\rangle \in \langle\langle Q, P\rangle\rangle$ is reachable, then for every $s \in L_p$, $u \in V(s)$, $p_u = h(s/u)$, and label path π in $\hat{M}_q(s[u \leftarrow p_u])$, $\#_{\langle\langle Q,\{p_u\}\rangle\rangle}(\pi) \le B$.*

Proof. Since $\langle\langle q, p\rangle\rangle$ is reachable, there are $t \in T_\Sigma$, $v \in V(t)$, and $\rho \in V_{\langle\langle q,p\rangle\rangle}(\hat{M}_{q_0}(t[v \leftarrow p]))$. We may assume that $t/v = s$ and hence $t/vu = s/u$. By Lemma 4.2, $\hat{M}_{q_0}(t[vu \leftarrow p_u]) = \hat{M}_{q_0}(t[v \leftarrow p])[\![\ldots]\!]$ with $[\![\ldots]\!] = [\![\langle\langle q', p\rangle\rangle \leftarrow \hat{M}_{q'}(s[u \leftarrow p_u]) \mid q' \in Q]\!]$. Clearly, $\mathrm{lpath}(\hat{M}_{q_0}(t[v \leftarrow p]), \rho) = w\langle\langle q, p\rangle\rangle$ for some $w \in (\langle\langle Q, \{p\}\rangle\rangle \cup \Delta)^*$. Since M is nondeleting (and hence so is $\hat{M}$), the substitution $[\![\ldots]\!]$ is nondeleting by Lemma 3.11(1), and thus, by Lemma 2.3(ii), there is a $w' \in (\langle\langle Q, \{p_u\}\rangle\rangle \cup \Delta)^*$ such that $w'\pi$ is a label path in $\hat{M}_{q_0}(t[v \leftarrow p])[\![\ldots]\!]$, i.e., in $\hat{M}_{q_0}(t[vu \leftarrow p_u])$. Now, $\#_{\langle\langle Q,\{p_u\}\rangle\rangle}(\pi) \le \#_{\langle\langle Q,\{p_u\}\rangle\rangle}(w'\pi)$, which is $\le B$, because B is a nesting bound for M. ∎

Consider a nondeleting $\mathrm{MTT}^{\mathrm{R}}$ M and an input tree $s \in T_\Sigma$. In section 6 we will often be interested in the part of s that lies between two nodes u and v of s, where v is a descendant of u; this part can be represented by the tree $s/u[v' \leftarrow p_v]$, where $v = uv'$ and $p_v = h(s/v)$. The shaded region in Figure 4.1 shows such a part of s.

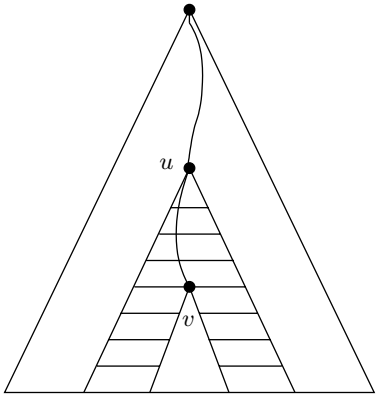


FIG. 4.1. *The tree s with shaded part $s/u[v' \leftarrow p_v]$.*

In particular, in section 6.2, we will need to know, if a state q of M processes this part, how many times the node v' is processed by a state q', i.e., how many times $\langle\langle q', p_v\rangle\rangle$ occurs in the tree $\hat{M}_q(s/u[v' \leftarrow p_v])$. If M is nondeleting and w is a node between u and v, i.e., a descendant of u and ancestor of v, then a lower bound for this number is given by summing for all states r the product of the number of occurrences of $\langle\langle r, p_w\rangle\rangle$ in $\hat{M}_q(s/u[w' \leftarrow p_w])$ and $\#_{\langle\langle q', p_v\rangle\rangle}(\hat{M}_r(s/w[v'' \leftarrow p_v]))$, where $v = wv''$. This is intuitively true because, due to nondeletion, for each occurrence of $\langle\langle r, p_w\rangle\rangle$ in $\hat{M}_q(s/u[w' \leftarrow p_w])$ there is in $\hat{M}_q(s/u[v' \leftarrow p_v])$ at least one occurrence of the tree $\hat{M}_r(s/w[v'' \leftarrow p_v])$ (without the parameters), and, due to parameter copying, there could be more than one such occurrence. This is stated in part (i) of the following lemma. Part (ii) of the lemma considers the case that M is fnest and fcp; then we can also give an upper bound for the number of occurrences of $\langle\langle q', p_v\rangle\rangle$ in $\hat{M}_q(s/u[v' \leftarrow p_v])$, because each occurrence of $\langle\langle r, p_w\rangle\rangle$ in $\hat{M}_q(s/u[w' \leftarrow p_w])$ can only be copied a bounded number of times.

LEMMA 4.14. *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be a nondeleting MTT^R. Let $q, q' \in Q$, $s \in T_\Sigma$, and $u, w, v \in V(s)$ such that u is an ancestor of w and w is an ancestor of v, i.e., $w = uw'$ and $v = wv''$ for some $w', v'' \in \mathbb{N}^*$, and let $v' = w'v''$, $p_w = h(s/w)$, and $p_v = h(s/v)$. Finally, let*

$$S = \sum_{r \in Q} \#_{\langle\langle q', p_v\rangle\rangle}(\hat{M}_r(s/w[v'' \leftarrow p_v])) \cdot \#_{\langle\langle r, p_w\rangle\rangle}(\hat{M}_q(s/u[w' \leftarrow p_w])).$$

Then the following two statements hold:

(i) $\#_{\langle\langle q', p_v\rangle\rangle}(\hat{M}_q(s/u[v' \leftarrow p_v])) \geq S$.

(ii) *If M is fnest and fcp with nesting bound B and parameter copying bound N, and $\langle\langle q, h(s/u)\rangle\rangle$ is reachable, then $\#_{\langle\langle q', p_v\rangle\rangle}(\hat{M}_q(s/u[v' \leftarrow p_v])) \leq N^B \cdot S$.*

Proof. Note that for $s' = s/u[v' \leftarrow p_v]$, $\hat{h}(s'/w') = p_w$, $s'[w' \leftarrow p_w] = s/u[w' \leftarrow p_w]$, and $s'/w' = s/w[v'' \leftarrow p_v]$. Hence, by Lemma 4.2 applied to s' and w', $\hat{M}_q(s/u[v' \leftarrow p_v]) = \xi[\![\ldots]\!]$, where $\xi = \hat{M}_q(s/u[w' \leftarrow p_w])$ and $[\![\ldots]\!] = [\![\langle\langle r, p_w\rangle\rangle \leftarrow \hat{M}_r(s/w[v'' \leftarrow p_v]) \mid r \in Q]\!]$, and thus, by Lemma 2.6,

$$(\times) \quad \#_{\langle\langle q', p_v\rangle\rangle}(\hat{M}_q(s/u[v' \leftarrow p_v])) = \sum_{\tilde{u} \in V_{\langle\langle r, p_w\rangle\rangle}(\xi), r \in Q} \#_{\langle\langle q', p_v\rangle\rangle}(\hat{M}_r(s/w[v'' \leftarrow p_v])) \prod F_{\xi,\tilde{u}}^{[\![\ldots]\!]}.$$

Since M is nondeleting, by Lemma 3.11(1), $\#_{y_j}(\hat{M}_{r'}(s/w[v'' \leftarrow p_v])) \geq 1$ for every $r' \in Q^{(m)}$ and $j \in [m]$. This implies that $\prod F_{\xi,\tilde{u}}^{[\![\ldots]\!]} \geq 1$. Thus, the sum in $(\times)$ is $\geq S$, because $|V_{\langle\!\langle r,p_w\rangle\!\rangle}(\xi)|$ equals $\#_{\langle\!\langle r,p_w\rangle\!\rangle}(\hat{M}_q(s/u[w' \leftarrow p_w]))$. This proves part (i).

For (ii), $\prod F_{\xi,\tilde{u}}^{[\![\ldots]\!]} \leq N^B$, because the number of elements of $\langle\!\langle Q, \{p_w\}\rangle\!\rangle$ that occur in $\mathrm{lpath}(\xi, \tilde{u})$ is $\leq B$ by Lemma 4.13 (using the assumption that $\langle\!\langle q, h(s/u)\rangle\!\rangle$ is reachable) and because, by Lemma 4.7, $\#_{y_j}(\hat{M}_{r'}(s/w[v'' \leftarrow p_v])) \leq N$ for every $r' \in Q^{(m)}$ and $j \in [m]$. Thus, the sum in $(\times)$ is $\leq N^B \cdot \sum_{\tilde{u} \in V_{\langle\!\langle r,p_w\rangle\!\rangle}(\xi), r \in Q} \#_{\langle\!\langle q',p_v\rangle\!\rangle}(\hat{M}_r(s/w[v'' \leftarrow p_v])) = N^B \cdot S$. □

Note that point (ii) of Lemma 4.14 can be strengthened by proving an upper bound of $N^{B-1} \cdot S$ for the number of occurrences of $\langle\!\langle q', p_v\rangle\!\rangle$ in $\hat{M}_q(s/u[v' \leftarrow p_v]))$. This is true because in $F_{\xi,\tilde{u}}^{[\![\ldots]\!]}$, the node $\tilde{u}$ itself (which is labeled by $\langle\!\langle r, p_w\rangle\!\rangle$ for some state r) is not taken into account; i.e., only proper ancestors of $\tilde{u}$ that are labeled by elements of $\langle\!\langle Q, \{p_w\}\rangle\!\rangle$ are counted; thus there are at most $B-1$ of them. We decided to leave out the "-1," because in the application of the lemma in the proof of Lemma 6.5 this will make the numbers more readable.

4.3. Finite copying implies linear size increase. In this subsection it is proved that if an $\mathrm{MTT^R}$ is finite copying, then it is lsi. Note that this result is not needed, because it follows from Lemma 4.9 (as discussed in the beginning of this section). The proof uses an intermediate, very natural notion, called *finite contribution*. Intuitively, an $\mathrm{MTT^R}$ M is of finite contribution if there is a bound c on the number of output nodes that are contributed by a node of the input tree. Clearly, if M is of finite contribution, then it is lsi (with bound c). Thus, in order to prove that finite copying implies lsi, it suffices to prove that if M is finite copying, then it is of finite contribution (Lemma 4.18). In fact, since one of the main results of this paper is that $\mathrm{MTT^R}$s of linear size increase realize the same class of translations as finite copying $\mathrm{MTT^R}$s (Theorem 7.2 and Lemma 4.9), it means that this is also the class of translations realized by $\mathrm{MTT^R}$s that are of finite contribution.

In order to compute the contribution by a node of the input tree s, we define an $\mathrm{MTT^R}$ M^s, which keeps in the label of each output node v the corresponding input node u that generated v. More precisely, if Δ is the output alphabet of M, then M^s has output alphabet $\langle \Delta, V(s)\rangle$, and the contribution by the node u of s is the number of symbols in $\langle \Delta, \{u\}\rangle$ that appear in $M_{q_0}^s(s')$, where s' is the "decorated version" of s; i.e., s' is obtained from s by changing, for every node w, its label σ into $\langle \sigma, w\rangle$.

Definition 4.15 (the $\mathrm{MTT^R}$ M^s, decorated version, contribution). *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R and $s \in T_\Sigma$. Then $M^s = (Q, P, \langle \Sigma, V(s)\rangle, \langle \Delta, V(s)\rangle, q_0, R^s, h^s)$ is the MTT^R such that for every $\langle \sigma, u\rangle \in \langle \Sigma, V(s)\rangle^{(k)}$, $k \geq 0$, and $p_1, \ldots, p_k \in P$,*

- $h^s_{\langle \sigma,u\rangle}(p_1, \ldots, p_k) = h_\sigma(p_1, \ldots, p_k)$, *and*
- $\mathrm{rhs}_{M^s}(q, \langle \sigma, u\rangle, \langle p_1, \ldots, p_k\rangle) = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)[\![\delta \leftarrow \langle \delta, u\rangle \mid \delta \in \Delta]\!]$.

The decorated version of s, *denoted by* $\mathrm{dec}(s)$, *is the unique tree in $T_{\langle \Sigma, V(s)\rangle}$ such that $V(\mathrm{dec}(s)) = V(s)$, and for every $u \in V(s)$, $\mathrm{dec}(s)[u] = \langle s[u], u\rangle$.*

For a node u of s, the set $V_{\langle \Delta,\{u\}\rangle}(M_{q_0}^s(\mathrm{dec}(s))) \subseteq V(M_{q_0}(s))$ is the set of output nodes contributed by u, *and the* contribution by u, *denoted by* $\mathrm{Contrib}_M(s, u)$, *is the cardinality $\#_{\langle \Delta,\{u\}\rangle}(M_{q_0}^s(\mathrm{dec}(s)))$ of this set.*

Note that every output node is contributed by a unique input node u (called its origin in [50]). Before we prove our first lemma about contribution, let us note some easy properties of the $\mathrm{MTT^R}$ M^s. Let $u \in V(s)$ and $q \in Q$.

(P1) $h^s(\mathrm{dec}(s)/u) = h(s/u)$.

(P2) For $s' \in T_{\langle\Sigma,V(s)\rangle}$, $\pi_\Delta(M^s_q(s')) = M_q(\pi_\Delta(s'))$, where π_Δ changes each symbol $\langle\delta,u\rangle$ into δ; i.e., it is the canonical projection from $\langle\Delta,V(s)\rangle$ to Δ. For $\hat{M}^s$ and $\hat{M}$ a similar statement holds.

Additionally, note the following two obvious facts about the projection π_Δ. Let Ω be a ranked alphabet disjoint with $\langle\Delta,V(s)\rangle$, $\xi \in T_{\Omega\cup\langle\Delta,V(s)\rangle}(Y)$, and $\xi' \in T_{\Omega\cup\langle\Delta,\{u\}\rangle}(Y)$. We assume that π_Δ is the identity on elements of Ω.

(D1) For $\beta \in (\Omega \cup Y)$, $V_\beta(\pi_\Delta(\xi)) = V_\beta(\xi)$.

(D2) For $\delta \in \Delta$, $V_\delta(\pi_\Delta(\xi')) = V_{\langle\delta,u\rangle}(\xi')$.

(P3) Let $P_0 = \{p^{(0)} \mid p \in P\}$.

(a) For $\xi \in T_{\langle\Sigma,V(s)\rangle}$, if $\#_{\langle\Sigma,\{u\}\rangle}(\xi) = 0$, then $\#_{\langle\Delta,\{u\}\rangle}(M^s_q(\xi)) = 0$.

(b) For $\xi \in T_{\langle\Sigma,V(s)\rangle\cup P_0}$, if $\#_{\langle\Sigma,\{u\}\rangle}(\xi) = 0$, then $\#_{\langle\Delta,\{u\}\rangle}(\hat{M}^s_q(\xi)) = 0$.

Let us prove property (P3) by induction on the structure of ξ. Let $\xi = \langle\sigma,v\rangle(\xi_1,\dots,\xi_k)$ with $\langle\sigma,v\rangle \in \langle\Sigma,V(s)\rangle^{(k)}$ and $k \geq 0$ such that $\#_{\langle\Sigma,\{u\}\rangle}(\xi) = 0$. By Lemma 3.5, $M^s_q(\xi) = \zeta[\![\langle q',x_i\rangle \leftarrow M^s_{q'}(\xi_i) \mid \langle q',x_i\rangle \in \langle Q,X_k\rangle]\!]$ with $\zeta = \mathrm{rhs}_{M^s}(q,\langle\sigma,v\rangle,\langle h^s(\xi_1),\dots,h^s(\xi_k)\rangle)$, and thus, by Lemma 2.6, $\#_{\langle\Delta,\{u\}\rangle}(M^s_q(\xi)) = S_1^{\langle\Delta,\{u\}\rangle} + S_2^{\langle\Delta,\{u\}\rangle}$, where $S_1^{\langle\Delta,\{u\}\rangle}$ and $S_2^{\langle\Delta,\{u\}\rangle}$ are the sums defined in that lemma (summed over all $\sigma \in \langle\Delta,\{u\}\rangle$). Now $S_1^{\langle\Delta,\{u\}\rangle} = 0$ because $V_{\langle\Delta,\{u\}\rangle}(\zeta) = \varnothing$ by the definition of the rules of M^s and by the fact that $v \neq u$ (because $\#_{\langle\Sigma,\{u\}\rangle}(\xi) = 0$). By induction, $\#_{\langle\Delta,\{u\}\rangle}(M^s_{q'}(\xi_i)) = 0$ and therefore also $S_2^{\langle\Delta,\{u\}\rangle} = 0$, which concludes the proof for the (a) case. For the (b) case the same proof holds, except that we have to consider the additional case $\xi = p \in P_0$: the right-hand side ζ of the p-rule of $\hat{M}^s$ is in $T_{\langle\langle Q,\{p\}\rangle\rangle}(Y)$ and thus $\#_{\langle\Delta,\{u\}\rangle}(\zeta) = 0$.

First, we want to present a lemma that computes, in the style of Lemma 2.6, the number $\mathrm{Contrib}_M(s,u)$ of output nodes contributed by u.

LEMMA 4.16. *Let $M = (Q,P,\Sigma,\Delta,q_0,R,h)$ be an MTT^R, $s \in T_\Sigma$, and $u \in V(s)$. Then*

$$\mathrm{Contrib}_M(s,u) = \sum_{\substack{v \in V_{\langle\langle q,p\rangle\rangle}(t)\\ q \in Q}} \sum_{w \in V_\Delta(\zeta_q)} \prod F^{[\![\text{-}]\!]}_{\zeta_q,w} \prod F^{[\![\dots]\!]}_{t,v}$$

with $p = h(s/u)$, $t = \hat{M}_{q_0}(s[u \leftarrow p])$, $\zeta_q = \mathrm{rhs}_M(q,\sigma,\langle p_1,\dots,p_k\rangle)$ for all $q \in Q$, where $\sigma = s[u] \in \Sigma^{(k)}$, $k \geq 0$, and $p_i = h(s/ui)$ for all $i \in [k]$, $[\![\text{-}]\!] = [\![\langle q',x_i\rangle \leftarrow M_{q'}(s/ui) \mid \langle q',x_i\rangle \in \langle Q,X_k\rangle]\!]$, and $[\![\dots]\!] = [\![\langle\langle q,p\rangle\rangle \leftarrow M_q(s/u) \mid q \in Q]\!]$.

Proof. By definition, $\mathrm{Contrib}_M(s,u) = \#_{\langle\Delta,\{u\}\rangle}(M^s_{q_0}(\mathrm{dec}(s)))$. Since, by the definition of dec, $\mathrm{dec}(s)[u] = \langle\sigma,u\rangle \in \langle\Sigma,V(s)\rangle^{(k)}$, we get by Lemmas 4.2 and 3.5 and property (P1), $M^s_{q_0}(\mathrm{dec}(s)) = t'[\![\mathrm{rhs}]\!][\![\text{-}]\!]'$, where $t' = \hat{M}^s_{q_0}(\mathrm{dec}(s)[u \leftarrow p])$, $[\![\mathrm{rhs}]\!] = [\![\langle\langle q,p\rangle\rangle \leftarrow \zeta'_q \mid q \in Q]\!]$ with $\zeta'_q = \mathrm{rhs}_{M^s}(q,\langle\sigma,u\rangle,\langle p_1,\dots,p_k\rangle)$ for $q \in Q$, and $[\![\text{-}]\!]' = [\![\langle q,x_i\rangle \leftarrow M^s_q(\mathrm{dec}(s)/ui) \mid \langle q,x_i\rangle \in \langle Q,X_k\rangle]\!]$. The application of Lemma 2.6 to $\#_{\langle\Delta,\{u\}\rangle}(t''[\![\text{-}]\!]')$ with $t'' = t'[\![\mathrm{rhs}]\!]$ gives $S_1 + S_2$, where $S_2 = 0$ because $\#_{\langle\Delta,\{u\}\rangle}(M^s_q(\mathrm{dec}(s)/ui)) = 0$ by property (P3)(a) and the fact that $\mathrm{dec}(s)/ui$ contains no symbol in $\langle\Sigma,\{u\}\rangle$ (by the definition of dec). Thus, $\mathrm{Contrib}_M(s,u) = S_1$, which equals

$$(*) \qquad \sum_{v \in V_{\langle\Delta,\{u\}\rangle}(t'[\![\mathrm{rhs}]\!])} \prod F^{[\![\text{-}]\!]'}_{t'[\![\mathrm{rhs}]\!],v}.$$

By the claim below, for $\Phi = [\![\mathrm{rhs}]\!]$ and $\Psi = [\![\text{-}]\!]'$, the sum in $(*)$ equals

$$\sum_{\gamma \in \langle\Delta,\{u\}\rangle} (S_1^\gamma + S_2^\gamma).$$

Now S_1^γ equals zero, because $V_{\langle\Delta,\{u\}\rangle}(t') = \varnothing$, which holds by property (P3)(b) and the fact that $\mathrm{dec}(s)[u \leftarrow p]$ contains no symbol in $\langle\Sigma,\{u\}\rangle$. Thus, the sum in $(*)$ equals $\sum_{\gamma\in\langle\Delta,\{u\}\rangle} S_2^\gamma =$

$$\sum_{\substack{v \,\in\, V_{\langle\langle q,p\rangle\rangle}(t') \\ q \,\in\, Q}} \; \sum_{w\in V_{\langle\Delta,\{u\}\rangle}(\zeta'_q)} \prod F^{[\![\text{-}]\!]'}_{\zeta'_q,w} \prod F^{[\![\cdots]\!]'}_{t',v},$$

where $[\![\ldots]\!]'$ is the substitution $[\![\langle\langle q,p\rangle\rangle \leftarrow M^s_q(\mathrm{dec}(s)/u) \mid q\in Q]\!]$. Let us now show that this sum equals the one of the lemma. For every $q\in Q$ it follows from property (D1) (for $\Omega = \langle\langle Q,\{p\}\rangle\rangle$ and $\beta = \langle\langle q,p\rangle\rangle$) that $V_{\langle\langle q,p\rangle\rangle}(t') = V_{\langle\langle q,p\rangle\rangle}(\pi_\Delta(t'))$, which equals $V_{\langle\langle q,p\rangle\rangle}(t)$ by (the $\hat{M}$-version of) property (P2), where π_Δ is the projection defined in that property. Since $\zeta'_q \in T_{\langle Q,X_k\rangle\cup\langle\Delta,\{u\}\rangle}(Y)$ it follows from property (D2) that $V_{\langle\Delta,\{u\}\rangle}(\zeta'_q) = V_\Delta(\pi_\Delta(\zeta'_q))$, which equals $V_\Delta(\zeta_q)$ because $\pi_\Delta(\zeta'_q) = \zeta_q$ by the definition of the rules of M^s. Now for $w\in V(\zeta'_q) = V(\zeta_q)$, $\prod F^{[\![\text{-}]\!]'}_{\zeta'_q,w} = \prod F^{[\![\text{-}]\!]}_{\zeta_q,w}$ because for $q'\in Q$, by (D1), $V_{\langle q',x_i\rangle}(\zeta'_q) = V_{\langle q',x_i\rangle}(\pi_\Delta(\zeta'_q))$, which equals $V_{\langle q',x_i\rangle}(\zeta_q)$, and for $y\in Y$, by (D1), $\#_y(M^s_q(\mathrm{dec}(s)/ui)) = \#_y(\pi_\Delta(M^s_q(\mathrm{dec}(s)/ui)))$, which equals $\#_y(M_q(s/ui))$ by (P2). Similarly, $\prod F^{[\![\cdots]\!]'}_{t',v} = \prod F^{[\![\cdots]\!]}_{t,v}$ for $v\in V(t') = V(t)$ because, as shown above, $V_{\langle\langle q,p\rangle\rangle}(t') = V_{\langle\langle q,p\rangle\rangle}(t)$ for $q\in Q$, and for $y\in Y$, by (D1), $\#_y(M^s_q(\mathrm{dec}(s)/u)) = \#_y(\pi_\Delta(M^s_q(\mathrm{dec}(s)/u)))$, which equals $\#_y(M_q(s))$ by (P2).

It remains to show the following claim, which is a generalization of Lemma 2.6 to two second-order tree substitutions Φ and Ψ. (More precisely, taking the substitution Ψ as the identity on $\Gamma - Y$ gives Lemma 2.6 for the case $\sigma = \gamma \notin \{\sigma_1,\ldots,\sigma_n\}\cup Y$.) Note that $\Phi\Psi$ denotes the composition of Ψ after Φ, i.e., $t(\Phi\Psi) = (t\Phi)\Psi$.

Claim. Let Γ be a ranked alphabet, $\Phi = [\![\sigma_i \leftarrow s_i \mid i\in[n]]\!]$ and $\Psi = [\![\tau_j \leftarrow \xi_j \mid j\in[m]]\!]$ second-order tree substitutions over Γ, and $t\in T_\Gamma$. For every $\gamma\in\Gamma - (\{\sigma_1,\ldots,\sigma_n,\tau_1,\ldots,\tau_m\}\cup Y)$,

$$(\times)\qquad \sum_{v\in V_\gamma(t\Phi)} \prod F^\Psi_{t\Phi,v} = S_1^\gamma + S_2^\gamma,$$

where

$$S_1^\gamma = \sum_{v\in V_\gamma(t)} \prod F^{\Phi\Psi}_{t,v} \qquad \text{and} \qquad S_2^\gamma = \sum_{\substack{v \,\in\, V_{\sigma_i}(t) \\ i \,\in\, [n]}} \; \sum_{w\in V_\gamma(s_i)} \prod F^\Psi_{s_i,w} \prod F^{\Phi\Psi}_{t,v}.$$

Proof of the claim. Note that the statement does not depend on the numbers $\#_\gamma(\xi_j)$. This is true because the substitution Ψ appears only in the Fs. In fact, for any node v of a tree ζ, $\prod F^\Psi_{\zeta,v} = \prod F^{\Psi'}_{\zeta,v}$ for every substitution $\Psi' = [\![\tau_j \leftarrow \xi'_j \mid j\in[m]]\!]$ with the property that $\#_y(\xi'_j) = \#_y(\xi_j)$ for every $y\in Y$ and $j\in[m]$; we denote this property by $E(\Psi,\Psi')$. For S_1^γ and S_2^γ a similar statement holds. (Note that if $E(\Psi,\Psi')$, then $E(\Phi\Psi,\Phi\Psi')$; this is true because, by associativity of second-order substitution, $\Phi\Psi = [\![\sigma_i \leftarrow s_i\Psi, \tau_j \leftarrow \xi_j \mid G]\!]$ and $\Phi\Psi' = [\![\sigma_i \leftarrow s_i\Psi', \tau_j \leftarrow \xi'_j \mid G]\!]$, where G denotes the statement "$i\in[n], j\in[m]$ with $\tau_j \notin \{\sigma_1,\ldots,\sigma_n\}$"; by the above, $E(\Psi,\Psi')$ implies that $\prod F^\Psi_{s_i,v} = \prod F^{\Psi'}_{s_i,v}$ for any node v of s_i, and thus for every $y\in Y$, $\sum_{v\in V_y(s_i)} \prod F^\Psi_{s_i,v} = \sum_{v\in V_y(s_i)} \prod F^{\Psi'}_{s_i,v}$, which means, by Lemma 2.6, that $\#_y(s_i\Psi) = \#_y(s_i\Psi')$.)

The idea of the proof is as follows. We will apply Lemma 2.6 twice: first to $\#_\gamma(t'\Psi')$, where $t' = t\Phi$ and Ψ' is a substitution with $E(\Psi,\Psi')$, and second to $\#_\gamma(tB)$

with $B = \Phi\Psi'$. The first application will give the left-hand side of the equation ($\times$), and the second one will give the right-hand side of that equation. Clearly, by definition of the composition of second-order tree substitutions, $\#_\gamma(t'\Psi') = \#_\gamma(tB)$.

Define $\Psi' = [\![\tau_j \leftarrow \xi'_j \mid j \in [m]]\!]$ with $E(\Psi, \Psi')$ and $\#_\gamma(\xi'_j) = 0$ for all $j \in [m]$. Then for $t' = t\Phi$, $\#_\gamma(t'\Psi')$ equals, by Lemma 2.6, $R_1^\gamma + R_2^\gamma$ with $R_1^\gamma = \sum_{v \in V_\gamma(t\Phi)} \prod F^{\Psi'}_{t\Phi,v}$ and $R_2^\gamma = 0$ because all the numbers $\#_\gamma(\xi'_j)$ are zero by the definition of Ψ'. Since $E(\Psi, \Psi')$, this means that $\#_\gamma(t'\Psi') = \sum_{v \in V_\gamma(t\Phi)} \prod F^{\Psi}_{t\Phi,v}$, which is the left-hand side of equation ($\times$).

By the associativity of second-order tree substitution, $B = \Phi\Psi'$ equals

$$[\![\sigma_i \leftarrow s_i\Psi', \tau_j \leftarrow \xi'_j \mid i \in [n], j \in [m] \text{ with } \tau_j \notin \Sigma_n]\!],$$

where $\Sigma_n = \{\sigma_1, \ldots, \sigma_n\}$. The application of Lemma 2.6 to $\#_\gamma(tB)$ gives $R_1^\gamma + R_2^\gamma$ with $R_1^\gamma = \sum_{v \in V_\gamma(t)} \prod F^{\Phi\Psi'}_{t,v}$ and

$$R_2^\gamma = \sum_{v \in V_{\sigma_i}(t), i \in [n]} \#_\gamma(s_i\Psi') \cdot \prod F^{\Phi\Psi'}_{t,v} + \sum_{v \in V_{\tau_j}(t), j \in [m], \tau_j \notin \Sigma_n} \#_\gamma(\xi'_j) \cdot \prod F^{\Phi\Psi'}_{t,v}.$$

Since $\#_\gamma(\xi'_j) = 0$, the second term of R_2^γ equals zero. In the first term of R_2^γ we apply Lemma 2.6 to $\#_\gamma(s_i\Psi')$, which gives $T_1^\gamma + T_2^\gamma$, where $T_2^\gamma = 0$ because $\#_\gamma(\xi'_j) = 0$, and $T_1^\gamma = \sum_{v \in V_{\sigma_i}(t), i \in [n]} \sum_{w \in V_\gamma(s_i)} \prod F^{\Psi'}_{s_i,w} \prod F^{\Phi\Psi'}_{t,v}$. Since $E(\Psi, \Psi')$, $R_1^\gamma = S_1^\gamma$ and $T_1^\gamma = S_2^\gamma$, which concludes the proof of the claim. □

Using Lemma 4.16 we can now prove that if an MTT$^{\mathrm{R}}$ is finite copying, then it is of finite contribution, which is defined next.

Definition 4.17 (finite contribution). *Let M be an MTT^R with input alphabet Σ. Then M is of* finite contribution *if there is a $c \in \mathbb{N}$ such that* $\mathrm{Contrib}_M(s, u) \leq c$ *for every $s \in T_\Sigma$ and $u \in V(s)$.*

Consider now a finite copying MTT$^{\mathrm{R}}$ M. In the translations of M, every node of the input tree is translated at most $I \cdot N^{I-1}$ times (cf. the discussion on page 71 of [19]), where I and N are input and parameter copying bounds for M, respectively. This implies that the number $\mathrm{Contrib}_M(s, u)$ of output nodes contributed by the node u is bounded.

Lemma 4.18. *Let M be an MTT^R. If M is finite copying, then it is of finite contribution.*

Proof. Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$, $s \in T_\Sigma$, and $u \in V(s)$. Let I be an input copying bound for M and let N be a parameter copying bound for M. Furthermore, let m be the maximal size of the right-hand side of a rule of M. By the definition of fci it follows that for $t = \hat{M}_{q_0}(s[u \leftarrow p])$ and $p = h(s/u)$, $\#_{\langle\langle Q, \{p\}\rangle\rangle}(t) \leq I$. By the definition of fcp it follows that, for every $v \in V_{\langle\langle q,p\rangle\rangle}(t)$ and $q \in Q$, $\prod F^{[\![\ldots]\!]}_{t,v} \leq N^{I-1}$, where $[\![\ldots]\!] = [\![\langle\langle q, p\rangle\rangle \leftarrow M_q(s/u) \mid q \in Q]\!]$. By Lemma 4.16 this means that $\mathrm{Contrib}_M(s, u) \leq I \cdot N^{I-1} \cdot \max\{\sum_{w \in V_\Delta(\zeta_q)} \prod F^{[\![\_]\!]}_{\zeta_q,w} \mid q \in Q\}$, where $[\![\_]\!] = [\![\langle q', x_i\rangle \leftarrow M_{q'}(s/ui) \mid \langle q', x_i\rangle \in \langle Q, X_k\rangle]\!]$. By the definition of m this is $\leq I \cdot N^{I-1} \cdot m \cdot \max\{\prod F^{[\![\_]\!]}_{\zeta_q,w} \mid q \in Q, w \in V_\Delta(\zeta_q)\} \leq I \cdot N^{I-1} \cdot m \cdot N^{m-1}$. Hence $\mathrm{Contrib}_M(s, u) \leq c$ for $c = I \cdot N^{I-1} \cdot m \cdot N^{m-1}$. □

As discussed in the beginning of this subsection, if an MTT$^{\mathrm{R}}$ is of finite contribution, then it is lsi. This holds because, by (P2), $\mathrm{size}(M_{q_0}(s)) = \mathrm{size}(M^s_{q_0}(\mathrm{dec}(s)))$ $= \sum_{u \in V(s)} \mathrm{Contrib}_M(s, u) \leq c \cdot \mathrm{size}(s)$. Together with Lemma 4.18 this gives us the desired result: finite copying implies lsi.

Theorem 4.19. *If an MTT^R is finite copying, then it is of linear size increase.*

5. Proper normal form. In section 4.3 we showed that if an $\mathrm{MTT}^{\mathrm{R}}$ is finite copying, then it is lsi. In sections 6 and 7 we want to prove that the converse also holds, i.e., that lsi implies finite copying. However, in general this does *not* hold: there are lsi $\mathrm{MTT}^{\mathrm{R}}$s that are not finite copying. Roughly speaking, the reason for this is that the part of the output tree of which unboundedly many copies are generated, by means of input variables or parameters, might be a fixed tree that does not change for different inputs. So, it can happen that an input tree s_n of size n generates a state sequence of length n, but the number of different output trees that are eventually generated by the states in the state sequence is finite. Then the $\mathrm{MTT}^{\mathrm{R}}$ is not finite copying in the input, but the translation it realizes can still be lsi (cf. the $\mathrm{MTT}^{\mathrm{R}}$ M at the beginning of section 5.1). Similarly, a tree $M_q(s_n)$ might contain n copies of a parameter y_j, but there are only finitely many different output trees that will be substituted for y_j in the actual output $M_{q_0}(s)$. Then M is not finite copying in the parameters, but the translation it realizes can be lsi (cf. the $\mathrm{MTT}^{\mathrm{R}}$ at the beginning of section 5.2).

Intuitively it should be clear that a state that generates, for any input, only a finite number of different output trees t is not needed; it can be eliminated by immediately substituting the correct tree t, which can be determined by regular look-ahead. This gives rise to a normal form, called *input proper*, which is treated in section 5.1. Similarly for a parameter y_j of a state q, if the number of actual output trees t that will be substituted for y_j is finite, then this parameter is not needed; it can be eliminated by immediately substituting the correct t, which can be computed in the states of the $\mathrm{MTT}^{\mathrm{R}}$. This gives rise to a normal form, called *parameter proper*; it is treated in section 5.2.

Altogether, an $\mathrm{MTT}^{\mathrm{R}}$ will be called *proper* if it is input proper, parameter proper, and productive. Again, this is a normal form; i.e., for every $\mathrm{MTT}^{\mathrm{R}}$ there is an equivalent one which is proper. Then in section 6 it can be proved that if a proper $\mathrm{MTT}^{\mathrm{R}}$ is lsi, then it is finite copying.

5.1. Input proper. Consider the following $\mathrm{MTT}^{\mathrm{R}}$ M, which is lsi but *not* fci. Let $M = (Q, \Sigma, \Delta, q_0, R)$ with $Q = \{q_0^{(0)}, q^{(0)}, q'^{(0)}\}$, $\Sigma = \{\gamma^{(1)}, a^{(0)}, b^{(0)}\}$, $\Delta = \{\sigma^{(2)}, a^{(0)}, b^{(0)}\}$, and R consisting of the following rules:

$$\begin{array}{lcl} \langle q_0, \gamma(x_1)\rangle & \to & \sigma(\langle q, x_1\rangle, \langle q', x_1\rangle), \\ \langle q, \gamma(x_1)\rangle & \to & \langle q, x_1\rangle, \\ \langle q', \gamma(x_1)\rangle & \to & \sigma(\langle q, x_1\rangle, \langle q', x_1\rangle), \\ \langle r, \alpha\rangle & \to & \alpha \quad (\text{for every } r \in Q \text{ and } \alpha \in \{a, b\}). \end{array}$$

Note that M is in fact a top-down tree transducer. Intuitively, M translates every monadic tree $s_n = \gamma(\dots\gamma(\alpha)\dots) = \gamma^n(\alpha)$ of height n (with $\alpha \in \{a, b\}$) into a comb $t_n = \sigma(\alpha, \sigma(\alpha, \dots\sigma(\alpha, \alpha)\dots))$ of height n. Thus, $\mathrm{size}(\tau_M(s)) \le 2 \cdot \mathrm{size}(s)$ for every $s \in T_\Sigma$ and so M is lsi. Clearly, M is not fci because $\mathrm{sts}_M(s_n, u) = q^n q'$ for $n \ge 1$ and $u = 1^n$ the unique leaf of s_n. The reason for this is that M generates n copies of q, but q generates only a finite number of different trees (viz. the trees a and b). How can we change M into an equivalent $\mathrm{MTT}^{\mathrm{R}}$ which is fci? The idea is to simply delete the state q and to determine by regular look-ahead the appropriate tree in $\{a, b\}$. In this example we just need $L_p = \{\gamma^n(a) \mid n \ge 0\}$ and $L_{p'} = \{\gamma^n(b) \mid n \ge 0\}$, and then the q_0-rule of M is replaced by two q_0-rules with right-hand sides $\sigma(a, \langle q', x_1\rangle)$ and $\sigma(b, \langle q', x_1\rangle)$ for look-ahead p and p', respectively, and similarly for the q'-rule.

We will say that an $\mathrm{MTT}^{\mathrm{R}}$ M is "input proper" if every state, except possibly the initial one, produces infinitely many output trees (in $T_\Delta(Y)$). More precisely, for

every look-ahead state p of M and every state $q \neq q_0$, M should produce infinitely many output trees taking L_p (the trees for which the look-ahead automaton arrives in state p) as input; in fact, this is only required if $\langle\langle q, p\rangle\rangle$ is reachable, i.e., if $\langle\langle q, p\rangle\rangle$ occurs in $\hat{M}_{q_0}(s[u \leftarrow p])$ for some s and u (see Definition 4.12). Note that q_0 is the only state that *may* generate only finitely many output trees; this is to make sure that an $\mathrm{MTT}^{\mathrm{R}}$ that generates only finitely many output trees can be transformed into one that is input proper.

The notion of input properness was defined in [1] for generalized syntax-directed translation schemes (which are a variant of top-down tree transducers) and was there called "reduced." We add two useful technical properties to it.

DEFINITION 5.1 (input proper). *An MTT^R $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ is* input proper *(for short, i-proper) if*

(i) *for every $q \in Q$ and $p \in P$ such that $q \neq q_0$ and $\langle\langle q, p\rangle\rangle$ is reachable, the set $\mathrm{Out}(q, p) = \{M_q(s) \mid s \in L_p\}$ is infinite;*
(ii) *q_0 does not occur in the right-hand sides of the rules in R; and*
(iii) *$L_p \neq \varnothing$ for every $p \in P$.*

Note that $\mathrm{Out}(q, p) \subseteq T_\Delta(Y_m)$ for $q \in Q^{(m)}$. Before it is proved (in Lemma 5.4) that i-properness is a normal form for $\mathrm{MTT}^{\mathrm{R}}$s, we need the following two straightforward lemmas about finiteness of $\mathrm{Out}(q, p)$.

LEMMA 5.2. *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R. For given $q \in Q^{(m)}$ and $p \in P$ it is decidable whether or not $\mathrm{Out}(q, p)$ is finite. Moreover, $\mathrm{Out}(q, p)$ can be constructed if it is finite.*

Proof. Let $\overline{M}$ be the $\mathrm{MTT}^{\mathrm{R}}$ constructed in the proof of Lemma 3.7. Then, for every $s \in T_\Sigma$, $\tau_{\overline{M}}(\overline{q}(s)) = M_q(s)[y_j \leftarrow \overline{y}_j \mid j \in [m]]$ and hence $M_q(s) = \tau_{\overline{M}}(\overline{q}(s))\Pi$, where $\Pi = [\overline{y}_j \leftarrow y_j \mid j \in [m]]$. The substitution Π can be realized by a (very simple) top-down tree transducer. Thus, for the regular tree language $L = \{\overline{q}(s) \mid s \in L_p\}$, $\mathrm{Out}(q, p) = \{M_q(s) \mid s \in L_p\} = \{\tau_{\overline{M}}(s)\Pi \mid s \in L\} = \tau_N(\tau_{\overline{M}}(L))$. By Lemma 3.8 the finiteness of $\tau_N(\tau_{\overline{M}}(L))$ is decidable, and in case of finiteness $\tau_N(\tau_{\overline{M}}(L))$ can be constructed. □

LEMMA 5.3. *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be a nondeleting MTT^R. Let $q \in Q$, $\sigma \in \Sigma^{(k)}$, $k \geq 1$, and $p, p_1, \ldots, p_k \in P$ such that $p = h_\sigma(p_1, \ldots, p_k)$ and $L_{p_j} \neq \varnothing$ for every $j \in [k]$. If $\langle r, x_i\rangle \in \langle Q, X_k\rangle$ occurs in $\mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)$ and $\mathrm{Out}(q, p)$ is finite, then $\mathrm{Out}(r, p_i)$ is finite.*

Proof. For $j \in [k] - \{i\}$ fix trees $s_j \in T_\Sigma$ with $h(s_j) = p_j$. Let $\xi = \zeta[\![\ldots]\!]$ with $\zeta = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)$ and $[\![\ldots]\!] = [\![\langle q', x_j\rangle \leftarrow M_{q'}(s_j) \mid q' \in Q, j \in [k] - \{i\}]\!]$. By the definition of $\mathrm{Out}(q, p)$, Lemma 3.5, and associativity of second-order tree substitution, $O = \{M_q(\sigma(s_1, \ldots, s_k)) \mid s_i \in L_{p_i}\} = \{\xi[\![s_i]\!] \mid s_i \in L_{p_i}\}$, where $[\![s_i]\!]$ denotes the substitution $[\![\langle q', x_i\rangle \leftarrow M_{q'}(s_i) \mid q' \in Q]\!]$ is a subset of $\mathrm{Out}(q, p)$ and hence finite. Since M is nondeleting, both $[\![\ldots]\!]$ and $[\![s_i]\!]$ are nondeleting by Lemma 3.11(1). Hence, by Lemma 2.1, ξ has a subtree $\langle r, x_i\rangle(\xi_1, \ldots, \xi_m)$, where $m = \mathrm{rank}_Q(r)$. Again by Lemma 2.1, $\xi[\![s_i]\!]$ has a subtree $\langle r, x_i\rangle(\xi_1, \ldots, \xi_m)[\![s_i]\!] = M_r(s_i)[y_j \leftarrow \xi_j[\![s_i]\!] \mid j \in [m]]$. Thus, for every $t \in \mathrm{Out}(r, p_i)$ (i.e., $t = M_r(s_i)$ for some $s_i \in L_{p_i}$) the tree $t[y_j \leftarrow \xi_j[\![s_i]\!] \mid j \in [m]]$ is a subtree of $\xi[\![s_i]\!]$, i.e., it is a subtree of a tree in the finite set O. This implies finiteness of $\mathrm{Out}(r, p_i)$. □

We are now ready to prove that i-properness is a normal form. The construction involved is similar to the one of Lemma 5.5 of [1] except that we apply it repeatedly to obtain an i-proper $\mathrm{MTT}^{\mathrm{R}}$ as opposed to their single application, which is insufficient (also in their formalism, which means that their proof of the lemma is incorrect).

LEMMA 5.4. *For every MTT^R M there is (effectively) an i-proper and productive*

MTT^R M' equivalent to M. If M is a T^R, then so is M'.

Proof. Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT$^{\mathrm{R}}$. By Lemma 3.10 we may assume that M is productive. Moreover, we may assume that q_0 does not occur in the right-hand side of any rule of M (if it does, replace it in all rules by a new state q_0' which has the same rules as q_0).

Before we construct the MTT$^{\mathrm{R}}$ M' which is i-proper and realizes the same translation as M, let us define an auxiliary notion. For each $p \in P$, let F_p denote the set $\{q \in Q \mid \mathrm{Out}(q,p) \text{ is finite}\}$ of states which produce finitely many output trees in $T_\Delta(Y)$ on input trees in L_p. Note that F_p can be constructed effectively, because, by Lemma 5.2, it is decidable whether or not $\mathrm{Out}(q,p)$ is finite. Moreover, $\mathrm{Out}(q,p)$ can be constructed for every $q \in F_p$.

The MTT$^{\mathrm{R}}$ M' is constructed in such a way that if $\langle r, x_i\rangle$ occurs in $\mathrm{rhs}_{M'}(q, \sigma, \langle p_1, \ldots, p_k\rangle)$, then $r \notin F_{p_i}$. This implies point (i) of i-properness of M' as follows: If $\langle\langle r, p\rangle\rangle \in \langle\langle Q, P\rangle\rangle$ is reachable (with $r \neq q_0$), then there are $s \in T_\Sigma$ and $u \in V(s)$ such that $\langle\langle r, p\rangle\rangle$ occurs in $\hat{M}'_{q_0}(s[u \leftarrow p])$. Since $r \neq q_0$, $u = vi$ for some $i \geq 1$ and $v \in \mathbb{N}^*$. By Lemma 4.3 this implies that $\langle r, x_i\rangle$ occurs in the right-hand side of a rule of M' with $p_i = p$. This means that $r \notin F_p$, i.e., $\mathrm{Out}(r,p)$ is infinite.

Note that an example for the construction in this proof can be found in Example 5.5.

We first construct the MTT$^{\mathrm{R}}$ $\pi(M)$ by simply deleting occurrences of $\langle r, x_i\rangle$ with $r \in F_{p_i}$ and replacing them by the correct tree in $\mathrm{Out}(r, p_i)$, which is determined by regular look-ahead. Due to the change of look-ahead automaton, an occurrence of $\langle r, x_i\rangle$ in the $(q, \sigma, \langle p_1, \ldots, p_k\rangle)$-rule of M with $r \notin F_{p_i}$ might produce only finitely many trees for the new look-ahead states (p_i, φ_i). For this reason we have to iterate the application of π until the sets F_p do not change anymore. This results in the desired MTT$^{\mathrm{R}}$ M'.

For each $p \in P$ let Φ_p be the (finite) set of all mappings $\varphi : F_p \to T_\Delta(Y)$ such that there is an $s \in L_p$ with $\varphi(q) = M_q(s)$ for every $q \in F_p$. Note that Φ_p is finite because $\varphi(q) \in \mathrm{Out}(q,p)$, which is finite for $q \in F_p$. This also implies that Φ_p can be obtained effectively by checking, for the (finitely many) mappings $\varphi : F_p \to \bigcup_{q \in F_p} \mathrm{Out}(q,p)$, whether or not φ is in Φ_p. This is decidable because $\varphi \in \Phi_p$ if and only if $K_{p,\varphi} = L_p \cap \bigcap_{q \in F_p} M_q^{-1}(\{\varphi(q)\})$ is nonempty; $K_{p,\varphi}$ is regular by Lemma 3.7 (and the closure of the regular tree languages under intersection) and hence has a decidable emptiness problem (cf., e.g., Theorem II.10.2 of [31]). The mappings in Φ_p partition L_p into the sets $K_{p,\varphi}$, which can be determined by regular look-ahead.

We now construct the MTT$^{\mathrm{R}}$ $\pi(M) = (Q, P', \Sigma, \Delta, q_0, R', h')$ as follows. Let $P' = \{(p, \varphi) \mid p \in P, \varphi \in \Phi_p\}$. For $\sigma \in \Sigma^{(k)}$ and $(p_1, \varphi_1), \ldots, (p_k, \varphi_k) \in P'$ let, for every $q \in Q^{(m)}$, the rule

$$\langle q, \sigma(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to \zeta_q\Theta \quad \langle (p_1, \varphi_1), \ldots, (p_k, \varphi_k)\rangle$$

be in R', where $\zeta_q = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)$ and $\Theta = [\![\langle r, x_i\rangle \leftarrow \varphi_i(r) \mid r \in F_{p_i}, i \in [k]]\!]$, and let $h'_\sigma((p_1, \varphi_1), \ldots, (p_k, \varphi_k)) = (p, \varphi)$, where $p = h_\sigma(p_1, \ldots, p_k)$ and $\varphi = \{(q, \zeta_q\Theta) \mid q \in F_p, \zeta_q = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)\}$.

Before we prove that the look-ahead automaton of $\pi(M)$ is as desired, let us show that it is well defined, i.e., that $\varphi \in \Phi_p$. We must show that there is an $s \in L_p$ such that, for every $q \in F_p$, $\varphi(q) = M_q(s)$. Since $\varphi_i \in \Phi_{p_i}$ for $i \in [k]$, there are $s_i \in L_{p_i}$ such that $\varphi_i(r) = M_r(s_i)$ for all $i \in [k]$ and $r \in F_{p_i}$. Hence, for $q \in F_p$, $\varphi(q) = \zeta_q\Theta$ with $\zeta_q = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)$ and $\Theta = [\![\langle r, x_i\rangle \leftarrow M_r(s_i) \mid \langle r, x_i\rangle \in \langle F_{p_i}, X_k\rangle]\!]$. By

Lemma 5.3 and the definition of F_p, only $\langle r, x_i\rangle$ with $r \in F_{p_i}$ occur in ζ_q. Therefore we can extend Θ to all elements of $\langle Q, X_k\rangle$. By Lemma 3.5 we get $\varphi(q) = M_q(s)$ for $s = \sigma(s_1, \ldots, s_k)$. Since $p = h_\sigma(p_1, \ldots, p_k)$, $s \in L_p$.

Claim 1. Let $s \in T_\Sigma$. If $h'(s) = (p, \varphi)$, then $p = h(s)$ and $\varphi(q) = M_q(s)$ for every $q \in F_p$.

The proof is by induction on the structure of s. Let $s = \sigma(s_1, \ldots, s_k)$ with $s_1, \ldots, s_k \in T_\Sigma$ and $h'(s_i) = (p_i, \varphi_i) \in P'$ for $i \in [k]$. By definition, p equals $h_\sigma(p_1, \ldots, p_k) = h(s)$. For $q \in F_p$, $\varphi(q) = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)\Theta$. By induction, $\varphi_i(r) = M_r(s_i)$ for all $i \in [k]$ and $r \in F_{p_i}$. For the same reason as above we can extend Θ to all elements of $\langle Q, X_k\rangle$ to get $M_q(s)$.

This claim implies that $\pi(M)$ satisfies point (iii) of i-properness. In fact, if $(p, \varphi) \in P'$, then $\varphi \in \Phi_p$, and so there exists $s \in L_p$ such that $\varphi(q) = M_q(s)$ for every $q \in F_p$. Thus, by Claim 1, $h'(s) = (p, \varphi)$. Hence, $L_{(p,\varphi)} \neq \varnothing$.

The $\mathrm{MTT}^{\mathrm{R}}$ $\pi(M)$ realizes the same translation as M. This follows from Claim 2 for $q = q_0$.

Claim 2. For $q \in Q$ and $s \in T_\Sigma$, $\pi(M)_q(s) = M_q(s)$.

Again we prove this by induction on s. Let $s = \sigma(s_1, \ldots, s_k)$ with $s_1, \ldots, s_k \in T_\Sigma$ and $h'(s_i) = (p_i, \varphi_i) \in P'$ for $i \in [k]$. By the definition of the rules of $\pi(M)$ and by Lemma 3.5, $\pi(M)_q(s)$ equals $\mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)\Theta[\![\_]\!]$, where $[\![\_]\!] = [\![\langle q', x_i\rangle \leftarrow \pi(M)_{q'}(s_i) \mid \langle q', x_i\rangle \in \langle Q, X_k\rangle]\!]$. By Claim 1, Θ equals $[\![\langle r, x_i\rangle \leftarrow M_r(s_i) \mid r \in F_{p_i}, i \in [k]]\!]$, and by induction $[\![\_]\!] = [\![\langle q', x_i\rangle \leftarrow M_{q'}(s_i) \mid \langle q', x_i\rangle \in \langle Q, X_k\rangle]\!]$. Thus $\Theta[\![\_]\!] = [\![\_]\!]$ and we get $\mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)[\![\_]\!]$, which, by Lemma 3.5, equals $M_q(s)$.

The $\mathrm{MTT}^{\mathrm{R}}$ $\pi(M)$ is productive because M is productive and the application of Θ does not delete nodes. Formally, consider a right-hand side $\zeta_q\Theta$ of $\pi(M)$ with $\zeta_q = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)$, $q \in Q^{(m)}$, and $m \geq 0$. For every $r \in F_{p_i}$, $\varphi_i(r) = M_r(s)$ for some $s \in T_\Sigma$. Thus, by Lemma 3.11(1), $\#_{y_\nu}(\varphi_i(r)) \geq 1$ for every $\nu \in [\mathrm{rank}_Q(r)]$; i.e., the substitution Θ is nondeleting. Since, for $j \in [m]$, $\#_{y_j}(\zeta_q) \geq 1$ this implies, by Lemma 2.1, that $\#_{y_j}(\zeta_q\Theta) \geq 1$; i.e., $\pi(M)$ is nondeleting. Analogously, by Lemma 3.11(2), $\#_{y_\nu}(\varphi_i(r)) \notin Y$ for $r \in F_{p_i}$ and $\nu \in [\mathrm{rank}_Q(r)]$; i.e., the substitution Θ is nonerasing. Since, for $j \in [m]$, $\zeta_q \notin Y$ this implies, by Lemma 2.2, that $\zeta_q\Theta \notin Y$; i.e., $\pi(M)$ is nonerasing.

Since $\pi(M)$ has the same states as M, $\pi(M)$ is a T^{R} if M is.

We now discuss the reason for iterating π. Consider an occurrence of $\langle r, x_i\rangle$ in the right-hand side of a rule of $\pi(M)$. We know that $r \notin F_{p_i}$, because each such occurrence is removed by the substitution Θ in the definition of the rules of $\pi(M)$. Thus, $\mathrm{Out}(r, p_i)$ is infinite. However, through the new look-ahead, the set L_{p_i} is partitioned into sets $L_{(p_i,\varphi_i)}$, $\varphi_i \in \Phi_{p_i}$ (to see this, consider an $s \in L_{p_i}$; then, by Claim 1, $s \in L_{(p_i,\varphi_i)}$, where φ_i is defined as $\varphi_i(q) = M_q(s)$ for every $q \in F_{p_i}$). Thus, we merely know, by Claim 2, that the union of $\mathrm{Out}(r, (p_i, \varphi_i))$ for all $\varphi_i \in \Phi_{p_i}$ is infinite, but for a particular $\varphi_i \in \Phi_{p_i}$, $\mathrm{Out}(r, (p_i, \varphi_i))$ might be finite, which means that $\pi(M)$ is not i-proper (see Example 5.5).

Let us now show that the iterative application of π yields an i-proper $\mathrm{MTT}^{\mathrm{R}}$. In particular, we iterate the application of π until

$$(*) \qquad F_{(p,\varphi)} = F_p \quad \text{for every } (p, \varphi) \in P'.$$

It follows from $(*)$ that if $\langle r, x_i\rangle$ occurs in the right-hand side of a rule of $\pi(M)$, then by the definition of Θ, $r \notin F_{p_i}$, and hence by $(*)$, $r \notin F_{(p_i,\varphi_i)}$. Thus $(*)$ implies (point (i) of) i-properness of $\pi(M)$, as argued in the beginning of this proof.

It remains to show that after a finite number of applications of π, $(*)$ holds. Clearly, $F_p \subseteq F_{(p,\varphi)} \subseteq Q$, because $\mathrm{Out}(q, (p, \varphi)) \subseteq \mathrm{Out}(q, p)$, as argued above. Let

us first show that, for every $(p,\varphi) \in P'$, $F_{(p,\varphi)} = F_p$ implies that (after constructing $\pi(\pi(M))$) $F_{((p,\varphi),\varphi')} = F_{(p,\varphi)}$ for every $\varphi' \in \Phi_{(p,\varphi)}$. Let $\varphi' \in \Phi_{(p,\varphi)}$; i.e., there is an $s \in L_{(p,\varphi)}$ such that $\varphi'(q) = \pi(M)_q(s)$ for every $q \in F_{(p,\varphi)} = F_p$. Since, by Claims 1 and 2, $\pi(M)_q(s) = M_q(s) = \varphi(q)$ for every $q \in F_p$, it follows that $\varphi' = \varphi$. This means that $L_{((p,\varphi),\varphi')} = \{s \in L_{(p,\varphi)} \mid \pi(M)_q(s) = \varphi'(q) \text{ for all } q \in F_{(p,\varphi)}\}$ equals $\{s \in L_{(p,\varphi)} \mid M_q(s) = \varphi(q) \text{ for all } q \in F_p\} = L_{(p,\varphi)}$. This implies that $\mathrm{Out}(q,((p,\varphi),\varphi')) = \mathrm{Out}(q,(p,\varphi))$ and thus $F_{((p,\varphi),\varphi')} = \{q \in Q \mid \mathrm{Out}(q,((p,\varphi),\varphi')) \text{ is finite}\} = \{q \in Q \mid \mathrm{Out}(q,(p,\varphi)) \text{ is finite}\} = F_{(p,\varphi)}$.

Now, after at most $k = |Q|$ iterations of π, $(*)$ holds. Let $(\cdots((p,\varphi_1),\varphi_2)\ldots,\varphi_k)$ be denoted by $(p,\varphi_1,\ldots,\varphi_k)$. Then, for every look-ahead state $(p,\varphi_1,\ldots,\varphi_k)$ of $\pi^k(M)$, $F_{(p,\varphi_1,\ldots,\varphi_{k-1})} = F_{(p,\varphi_1,\ldots,\varphi_k)}$. This is true because $F_p = \varnothing$ implies $F_{(p,\varphi_1)} = \varnothing$ (since $\Phi_p = \{\varphi_1\}$), and $F_{(p,\varphi_1,\ldots,\varphi_i)} = F_{(p,\varphi_1,\ldots,\varphi_{i+1})}$ implies that $F_{(p,\varphi_1,\ldots,\varphi_j)} = F_{(p,\varphi_1,\ldots,\varphi_i)}$ for all $j \geq i$ (by the above). Since a sequence of nonempty subsets of Q in which each set is a proper subset of the next one has length at most $|Q| = k$, $F_{(p,\varphi_1,\ldots,\varphi_{k-1})} = F_{(p,\varphi_1,\ldots,\varphi_k)}$. Thus, $M' = \pi^k(M)$ is i-proper. □

The next example illustrates the construction of an i-proper $\mathrm{MTT}^{\mathrm{R}}$ following the proof of Lemma 5.4.

Example 5.5. For simplicity let us consider an $\mathrm{MTT}^{\mathrm{R}}$ without parameters, i.e., a T^{R}. Let $M = (Q,P,\Sigma,\Delta,q_0,R,h)$ be a T^{R} with $Q = \{q_0,q,q',i\}$, $P = \{p\}$, $\Sigma = \{\alpha^{(0)},\gamma^{(1)},\sigma^{(1)}\}$, $\Delta = \{\alpha^{(0)},\beta^{(0)},\gamma^{(1)},\sigma^{(1)},\delta^{(2)}\}$, and let R consist of the following rules:

$$\begin{array}{lcll}
\langle q_0,\gamma(x_1)\rangle & \to & \delta(\langle q,x_1\rangle,\langle i,x_1\rangle) & \langle p\rangle,\\
\langle q_0,\sigma(x_1)\rangle & \to & \langle q',x_1\rangle & \langle p\rangle,\\
\langle q,\gamma(x_1)\rangle & \to & \alpha & \langle p\rangle,\\
\langle q,\sigma(x_1)\rangle & \to & \beta & \langle p\rangle,\\
\langle q',\gamma(x_1)\rangle & \to & \alpha & \langle p\rangle,\\
\langle q',\sigma(x_1)\rangle & \to & \sigma(\langle i,x_1\rangle) & \langle p\rangle,\\
\langle i,\gamma(x_1)\rangle & \to & \gamma(\langle i,x_1\rangle) & \langle p\rangle,\\
\langle i,\sigma(x_1)\rangle & \to & \sigma(\langle i,x_1\rangle) & \langle p\rangle,\\
\langle r,\alpha\rangle & \to & \alpha & \text{for each } r \in Q.
\end{array}$$

Let us now define $M_1 = \pi(M) = (Q,P',\Sigma,\Delta,q_0,R',h')$. We obtain $F_p = \{q\}$ and $\Phi_p = \{\varphi_\alpha,\varphi_\beta\}$ with $\varphi_\alpha = \{(q,\alpha)\}$ and $\varphi_\beta = \{(q,\beta)\}$, and thus $P' = \{(p,\varphi_\alpha),(p,\varphi_\beta)\}$. As can easily be verified, the rules of the look-ahead automaton of M_1 are the following: $h'_\alpha = (p,\varphi_\alpha)$, $h'_\gamma((p,\varphi_\alpha)) = h'_\gamma((p,\varphi_\beta)) = (p,\varphi_\alpha)$, $h'_\sigma((p,\varphi_\alpha)) = h'_\sigma((p,\varphi_\beta)) = (p,\varphi_\beta)$.

The q-, q'-, and i-rules in R' are identical to the ones in R for both new look-ahead states. The q_0-rules in R' are the following:

$$\begin{array}{lcll}
\langle q_0,\gamma(x_1)\rangle & \to & \delta(\alpha,\langle i,x_1\rangle) & \langle(p,\varphi_\alpha)\rangle,\\
\langle q_0,\gamma(x_1)\rangle & \to & \delta(\beta,\langle i,x_1\rangle) & \langle(p,\varphi_\beta)\rangle,\\
\langle q_0,\sigma(x_1)\rangle & \to & \langle q',x_1\rangle & \langle(p,\varphi_\alpha)\rangle,\\
\langle q_0,\sigma(x_1)\rangle & \to & \langle q',x_1\rangle & \langle(p,\varphi_\beta)\rangle.
\end{array}$$

Note that $L_{(p,\varphi_\alpha)} = \{\alpha\} \cup \{\gamma(s) \mid s \in T_\Sigma\}$ and $L_{(p,\varphi_\beta)} = \{\sigma(s) \mid s \in T_\Sigma\}$. Hence $\mathrm{Out}(q',(p,\varphi_\alpha)) = \{\alpha\}$, and so the T^{R} M_1 is not i-proper yet, because $F_{(p,\varphi_\alpha)} = \{q,q'\} \neq F_p$. Thus we have to apply π again. Let $M' = \pi(M_1) = (Q,P'',\Sigma,\Delta,q_0,R'',h'')$. We get $\Phi_{(p,\varphi_\alpha)} = \{\varphi\}$, with $\varphi = \{(q,\alpha),(q',\alpha)\}$ and $\Phi_{(p,\varphi_\beta)} = \{\varphi_\beta\}$. Thus $P'' = \{((p,\varphi_\alpha),\varphi),((p,\varphi_\beta),\varphi_\beta)\}$. The look-ahead automaton of M' stays the same as for M_1 except for a renaming of states: (p,φ_α) becomes $((p,\varphi_\alpha),\varphi)$ and (p,φ_β)

becomes $((p,\varphi_\beta),\varphi_\beta)$. The q-, q'-, and i-rules in R'' are identical to the ones in R' (and R) for all look-ahead states. The q_0-rules in R'' are the following:

$$\begin{array}{lcll}
\langle q_0,\gamma(x_1)\rangle & \to & \delta(\alpha,\langle i,x_1\rangle) & \langle((p,\varphi_\alpha),\varphi)\rangle,\\
\langle q_0,\gamma(x_1)\rangle & \to & \delta(\beta,\langle i,x_1\rangle) & \langle((p,\varphi_\beta),\varphi_\beta)\rangle,\\
\langle q_0,\sigma(x_1)\rangle & \to & \alpha & \langle((p,\varphi_\alpha),\varphi)\rangle,\\
\langle q_0,\sigma(x_1)\rangle & \to & \langle q',x_1\rangle & \langle((p,\varphi_\beta),\varphi_\beta)\rangle.
\end{array}$$

The T^{R} M' is i-proper because $F_{((p,\varphi_\alpha),\varphi)}=\{q,q'\}=F_{(p,\varphi_\alpha)}$ and $F_{((p,\varphi_\beta),\varphi_\beta)}=\{q\}=F_{(p,\varphi_\beta)}$. We finally note that it is easy to transform M into a generalized syntax-directed translation scheme that forms a counterexample to the proof of Lemma 5.5 of [1].

5.2. Parameter proper. Consider the following MTT M which is lsi but *not* fcp. Let $M=(Q,\Sigma,\Delta,q_0,R)$ with $Q=\{q_0^{(0)},q^{(1)}\}$, $\Sigma=\{\sigma^{(2)},\gamma^{(2)},\alpha^{(0)},\beta^{(0)}\}$, and $\Delta=\{\sigma^{(2)},\gamma^{(2)},\alpha^{(1)},\beta^{(1)},\bar\sigma^{(0)},\bar\gamma^{(0)}\}$. For all $\delta\in\{\sigma,\gamma\}$ and $a\in\{\alpha,\beta\}$, let the following rules be in R:

$$\begin{array}{lcl}
\langle q_0,\delta(x_1,x_2)\rangle & \to & \delta(\langle q,x_1\rangle(\bar\delta),\langle q,x_2\rangle(\bar\delta)),\\
\langle q,\delta(x_1,x_2)\rangle(y_1) & \to & \delta(\langle q,x_1\rangle(y_1),\langle q,x_2\rangle(y_1)),\\
\langle q_0,a\rangle & \to & a(\bar a),\\
\langle q,a\rangle(y_1) & \to & a(y_1).
\end{array}$$

Intuitively, M moves the root symbol of the input tree to each of its leaves; e.g., for $s=\sigma(\gamma(\alpha,\beta),\alpha)$ we get $\tau_M(s)=\sigma(\gamma(\alpha(\bar\sigma),\beta(\bar\sigma),\alpha(\bar\sigma)))$. Thus, M is lsi (because $\mathrm{size}(\tau_M(s))\le 2\cdot\mathrm{size}(s)$). Clearly, M is not fcp, because $\#_{y_1}(M_q(s))$ equals the number of leaves of s. This time, the reason is that M generates a lot of parameter occurrences which have only finitely many "argument trees" (viz., $\bar\sigma$ and $\bar\gamma$). A jth argument tree for q and p is a tree ξ_j such that $\langle\!\langle q,p\rangle\!\rangle(\xi_1,\dots,\xi_m)$ is a subtree of some $\hat M_{q_0}(s[u\leftarrow p])$.

The idea of the next normal form is to eliminate parameters y_j of q for which there are only finitely many jth argument trees (for look-ahead p). This can be done by keeping the information on these argument trees in the states of the new $\mathrm{MTT}^{\mathrm{R}}$ and by appropriately replacing y_j by the correct argument tree in each right-hand side. For the example MTT M above, we have to add states $q_{\bar\delta}$, $\delta\in\{\sigma,\gamma\}$ of rank zero, and take as rules

$$\begin{array}{lcll}
\langle q_0,\delta(x_1,x_2)\rangle & \to & \delta(\langle q_{\bar\delta},x_1\rangle,\langle q_{\bar\delta},x_2\rangle), & \\
\langle q_{\bar\delta},\rho(x_1,x_2)\rangle & \to & \rho(\langle q_{\bar\delta},x_1\rangle,\langle q_{\bar\delta},x_2\rangle) & \text{for } \rho\in\{\sigma,\gamma\},\\
\langle q_0,a\rangle & \to & a(\bar a) & \text{for } a\in\{\alpha,\beta\},\\
\langle q_{\bar\delta},a\rangle & \to & a(\bar\delta) & \text{for } a\in\{\alpha,\beta\}.
\end{array}$$

This shows that the translation τ_M can actually be realized by a top-down tree transducer (which is in general, of course, not the case).

DEFINITION 5.6 (parameter proper, proper). *An MTT^R $M=(Q,P,\Sigma,\Delta,q_0,R,h)$ is* parameter proper *(for short, p-proper) if for every $q\in Q^{(m)}$, $m\ge 1$, $j\in[m]$, and $p\in P$*

(i) *if $\langle\!\langle q,p\rangle\!\rangle$ is reachable, then the set* $\mathrm{Arg}(q,j,p)$ *defined as*

$$\{t/vj \mid \exists s\in T_\Sigma, u\in V(s): t=\hat M_{q_0}(s[u\leftarrow p]), v\in V(t), t[v]=\langle\!\langle q,p\rangle\!\rangle\}$$

is infinite; and

(ii) *if* $\langle\!\langle q,p\rangle\!\rangle$ *is not reachable, then* $\#_{y_j}(M_q(s)) \leq 1$ *for all* $s \in L_p$.

The MTT^R M *is* proper *if it is productive and both i-proper and p-proper.*

Note that $\mathrm{Arg}(q,j,p) \subseteq T_{\langle\!\langle Q,\{p\}\rangle\!\rangle \cup \Delta}$. Note also that $\langle\!\langle q,p\rangle\!\rangle$ is reachable if and only if $\mathrm{Arg}(q,j,p) \neq \varnothing$.

Point (ii) in Definition 5.6 says that if a parameter appears more than once in $M_q(s)$, then $\langle\!\langle q, h(s)\rangle\!\rangle$ is reachable. This (mild) additional requirement is needed to force an lsi $\mathrm{MTT^R}$ to be fcp, because Definition 4.6 of the fcp property requires $\#_{y_j}(M_q(s)) \leq N$ for *all* states q; i.e., $\langle\!\langle q, h(s)\rangle\!\rangle$ might *not* be reachable.

Similar to the case of i-properness, we present two lemmas concerning the finiteness of $\mathrm{Arg}(q,j,p)$. First, let us show that it is decidable whether $\mathrm{Arg}(q,j,p)$ is infinite.

LEMMA 5.7. *Let* $M = (Q,P,\Sigma,\Delta,q_0,R,h)$ *be an* MTT^R. *For given* $q \in Q^{(m)}$, $m \geq 1$, $j \in [m]$, *and* $p \in P$, *it is decidable whether or not* $\mathrm{Arg}(q,j,p)$ *is finite. Moreover,* $\mathrm{Arg}(q,j,p)$ *can be constructed if it is finite.*

Proof. Let K_p be the regular tree language $\{s \in T_{\hat{\Sigma}} \mid p \text{ occurs exactly once in } s\}$ with $\hat{\Sigma} = \Sigma \cup \{p^{(0)}\}$. Then $\tau_{\hat{M}}(K_p) \subseteq T_{\langle\!\langle Q,\{p\}\rangle\!\rangle \cup \Delta}$. We now construct a partial nondeterministic top-down tree transducer N which takes a tree in $T_{\langle\!\langle Q,\{p\}\rangle\!\rangle \cup \Delta}$ as input and generates as output the jth subtree of an occurrence of $\langle\!\langle q,p\rangle\!\rangle$. (A partial nondeterministic top-down tree transducer is defined as in Definitions 3.1 and 3.2, but for q and σ there may be no or several rules of the form $\langle q,\sigma(x_1,\ldots,x_k)\rangle \to \zeta$.) Let $N = (\{r^{(0)},\mathrm{id}^{(0)}\},\Gamma,\Gamma,r,R')$, where $\Gamma = \langle\!\langle Q,\{p\}\rangle\!\rangle \cup \Delta$ and R' consists of the following rules:

$$\begin{array}{lcll}
\langle r,\gamma(x_1,\ldots,x_k)\rangle & \to & \langle r,x_i\rangle & \text{for all } \gamma \in \Gamma^{(k)}, k \geq 1, i \in [k],\\
\langle r,\langle\!\langle q,p\rangle\!\rangle(x_1,\ldots,x_m)\rangle & \to & \langle \mathrm{id},x_j\rangle, & \\
\langle \mathrm{id},\gamma(x_1,\ldots,x_k)\rangle & \to & \gamma(\langle \mathrm{id},x_1\rangle,\ldots,\langle \mathrm{id},x_k\rangle) & \text{for all } \gamma \in \Gamma^{(k)}, k \geq 0.
\end{array}$$

Clearly, $\tau_N(\tau_{\hat{M}}(K_p)) = \mathrm{Arg}(q,j,p)$, because every tree t in $\tau_{\hat{M}}(K_p)$ equals $\hat{M}_{q_0}(s[u \leftarrow p])$ for some s and u, and for every subtree $\langle\!\langle q,p\rangle\!\rangle(\xi_1,\ldots,\xi_m)$ of t, $(t,\xi_j) \in \tau_N$. The finiteness of $L = \tau_N(\tau_{\hat{M}}(K_p))$ can be decided by Lemma 3.8, and in case of finiteness L can be constructed. □

LEMMA 5.8. *Let* $M = (Q,P,\Sigma,\Delta,q_0,R,h)$ *be an i-proper and productive* MTT^R. *Let* $q \in Q^{(n)}$, $\sigma \in \Sigma^{(k)}$, $n,k \geq 0$, *and* $p,p_1,\ldots,p_k \in P$ *such that* $p = h_\sigma(p_1,\ldots,p_k)$ *and* $\langle\!\langle q,p\rangle\!\rangle$ *is reachable. Let* $\langle r,x_i\rangle(t_1,\ldots,t_m)$ *be a subtree of* $\mathrm{rhs}_M(q,\sigma,\langle p_1,\ldots,p_k\rangle)$ *with* $r \in Q^{(m)}$, $m \geq 0$, $i \in [k]$, *and* $t_1,\ldots,t_m \in T_{\langle Q,X_k\rangle \cup \Delta}(Y_n)$.

For $j \in [m]$, *the set* $\mathrm{Arg}(r,j,p_i)$ *is infinite if in* t_j *there is*

(i) *an occurrence of* $y_\mu \in Y_n$, *where* $\mathrm{Arg}(q,\mu,p)$ *is infinite, or*

(ii) *an occurrence of an element of* $\langle Q, X_k - \{x_i\}\rangle$, *or*

(iii) *an occurrence of* $y_\mu \in Y_n$ *such that there is a* $\xi \in \mathrm{Arg}(q,\mu,p)$ *for which* $\xi[\![\mathrm{rhs}]\!]$ *contains an occurrence of an element of* $\langle Q, X_k - \{x_i\}\rangle$, *where* $[\![\mathrm{rhs}]\!]$ *denotes the substitution* $[\![\langle\!\langle q',p\rangle\!\rangle \leftarrow \mathrm{rhs}_M(q',\sigma,\langle p_1,\ldots,p_k\rangle) \mid q' \in Q]\!]$.

Proof. Informally, and roughly speaking, the idea of this lemma is the following. Consider the given rule

$$\langle q,\sigma(x_1,\ldots,x_k)\rangle(y_1,\ldots,y_n) \to \ldots\langle r,x_i\rangle(t_1,\ldots,t_j,\ldots,t_m)\ldots \qquad \langle p_1,\ldots,p_k\rangle.$$

In case (i), if $\xi_\mu \in \mathrm{Arg}(q,\mu,p)$, i.e., ξ_μ is a μth argument tree of q and p, then, due to the application of the above rule, the tree $t_j[y_\mu \leftarrow \xi_\mu[\![\mathrm{rhs}]\!]]$ determines a jth argument tree of r and p_i, i.e., an element of $\mathrm{Arg}(r,j,p_i)$. By "determines" we mean that the remaining parameters and the input variables $\neq x_i$ have to be substituted by

appropriate trees, and then all $\langle q', s_\nu\rangle$ (where s_ν is substituted for $x_\nu \neq x_i$) have to be substituted by $M_{q'}(s_\nu)$, and $\langle q', x_i\rangle$ by $\langle\!\langle q', p_i\rangle\!\rangle$. Now, since it is given that there are infinitely many such ξ_μ, there are also infinitely many $t_j[y_\mu \leftarrow \xi_\mu[\![\mathrm{rhs}]\!]]$, and these determine infinitely many elements of $\mathrm{Arg}(r, j, p_i)$ because all involved substitutions are productive (see Lemma 2.7). In case (ii), if t_j contains an occurrence of $\langle q', x_\nu\rangle$ with $\nu \neq i$, and $t \in \mathrm{Out}(q', p_\nu)$, then the tree $t_j[\![\langle q', x_\nu\rangle \leftarrow t]\!]$ determines an element of $\mathrm{Arg}(r, j, p_i)$. Since M is i-proper, $\mathrm{Out}(q', p_\nu)$ is infinite and hence so is $\mathrm{Arg}(r, j, p_i)$. Finally, in case (iii), if $\xi[\![\mathrm{rhs}]\!]$ contains an occurrence of $\langle q', x_\nu\rangle$ with $\nu \neq i$, and $t \in \mathrm{Out}(q', p_\nu)$, then $t_j[y_\mu \leftarrow \xi[\![\mathrm{rhs}]\!][\![\langle q', x_\nu\rangle \leftarrow t]\!]]$ determines an element of $\mathrm{Arg}(r, j, p_i)$. We now turn to the formal proof.

Consider $s \in T_\Sigma$, $u \in V(s)$, and $\xi_1, \ldots, \xi_n \in T_{\langle\!\langle Q,\{p\}\rangle\!\rangle \cup \Delta}$ such that $\langle\!\langle q, p\rangle\!\rangle(\xi_1, \ldots, \xi_n)$ is a subtree of $\hat{M}_{q_0}(s[u \leftarrow p])$. Consider also $s_\nu \in L_{p_\nu}$ for $\nu \in [k]$. Note that such trees exist because $\langle\!\langle q, p\rangle\!\rangle$ is reachable and because M satisfies point (iii) of i-properness.

Let $s' = s[u \leftarrow \sigma(s_1, \ldots, s_k)]$. Note that $s'/u = \sigma(s_1, \ldots, s_k)$ is in L_p and that $s'[u \leftarrow p] = s[u \leftarrow p]$. By Lemma 4.3, $\hat{M}_{q_0}(s'[ui \leftarrow p_i]) = \hat{M}_{q_0}(s[u \leftarrow p])[\![\mathrm{rhs}]\!]\Psi_{s_1,\ldots,s_k}[\![i]\!]$, with $[\![\mathrm{rhs}]\!]$ as in (iii), $\Psi_{s_1,\ldots,s_k} = [\![\langle q', x_\nu\rangle \leftarrow M_{q'}(s_\nu) \mid q' \in Q, \nu \in [k] - \{i\}]\!]$, and $[\![i]\!] = [\![\langle q', x_i\rangle \leftarrow \langle\!\langle q', p_i\rangle\!\rangle \mid q' \in Q]\!]$.

Since M is nondeleting, so is $[\![\mathrm{rhs}]\!]$ and, by Lemma 3.11(1), so is $\Psi_{s_1,\ldots,s_k}$. Then, by Lemma 2.1, the tree $\hat{M}_{q_0}(s'[ui \leftarrow p_i])$ has a subtree $\langle\!\langle q, p\rangle\!\rangle(\xi_1, \ldots, \xi_n)[\![\mathrm{rhs}]\!]\Psi_{s_1,\ldots,s_k}[\![i]\!]$ $= \zeta\Pi_{\xi_1,\ldots,\xi_n}\Psi_{s_1,\ldots,s_k}[\![i]\!]$ with $\zeta = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)$ and $\Pi_{\xi_1,\ldots,\xi_n} = [y_\eta \leftarrow \xi_\eta[\![\mathrm{rhs}]\!] \mid \eta \in [n]]$. Again by Lemma 2.1 it has a subtree $\langle\!\langle r, p_i\rangle\!\rangle(t'_1, \ldots, t'_m)$, where, for $j \in [m]$,

$$(*) \qquad t'_j = t_j\Pi_{\xi_1,\ldots,\xi_n}\Psi_{s_1,\ldots,s_k}[\![i]\!] \in \mathrm{Arg}(r, j, p_i).$$

(i) Let $j \in [m]$ such that y_μ is a subtree of t_j. By Lemma 2.1, $y_\mu\Pi_{\xi_1,\ldots,\xi_n}\Psi_{s_1,\ldots,s_k}[\![i]\!]$ $= \xi_\mu[\![\mathrm{rhs}]\!]\Psi_{s_1,\ldots,s_k}[\![i]\!]$ is a subtree of t'_j. Thus $\mathrm{size}(t'_j) \geq \mathrm{size}(\xi_\mu[\![\mathrm{rhs}]\!]\Psi_{s_1,\ldots,s_k}[\![i]\!])$, which is $\geq \mathrm{size}(\xi_\mu)$ by Lemma 2.7 and the fact that $[\![\mathrm{rhs}]\!]$ and $\Psi_{s_1,\ldots,s_k}$ are productive by Lemma 3.11. We now let $\xi_1, \ldots, \xi_n$ vary in $(*)$: For every ξ_μ in the infinite set $\mathrm{Arg}(q, \mu, p)$ there are $s \in T_\Sigma$, $u \in V(s)$, and ξ_η, $\eta \in [n] - \{\mu\}$ such that $\langle\!\langle q, p\rangle\!\rangle(\xi_1, \ldots, \xi_n)$ is a subtree of $\hat{M}_{q_0}(s[u \leftarrow p])$; then the size of $t_j\Pi_{\xi_1,\ldots,\xi_n}\Psi_{s_1,\ldots,s_k}[\![i]\!] \in \mathrm{Arg}(r, j, p_i)$ is $\geq \mathrm{size}(\xi_\mu)$. Thus, $\mathrm{Arg}(r, j, p_i)$ is infinite.

(ii) Let $j \in [m]$, $q' \in Q^{(l)}$, $l \geq 0$, and $\nu \in [k] - \{i\}$ such that t_j has a subtree $\langle q', x_\nu\rangle(\bar{t}_1, \ldots, \bar{t}_l)$ for some trees $\bar{t}_1, \ldots, \bar{t}_l$. Then $\langle\!\langle q', p_\nu\rangle\!\rangle$ is reachable by the same argument as given above equation $(*)$ (where we showed that $\langle\!\langle r, p_i\rangle\!\rangle$ is reachable). By Lemma 2.1, t'_j has the subtree $M_{q'}(s_\nu)[y_\eta \leftarrow \bar{t}_\eta\Pi_{\xi_1,\ldots,\xi_n}\Psi_{s_1,\ldots,s_k}[\![i]\!] \mid \eta \in [l]]$, the size of which is $\geq \mathrm{size}(M_{q'}(s_\nu))$. Since M satisfies points (i) and (ii) of i-properness, the set $\mathrm{Out}(q', p_\nu) = \{M_{q'}(s_\nu) \mid s_\nu \in L_{p_\nu}\}$ is infinite. We now let s_ν vary in $(*)$: For every $s_\nu \in L_{p_\nu}$ the size of $t_j\Pi_{\xi_1,\ldots,\xi_n}\Psi_{s_1,\ldots,s_k}[\![i]\!] \in \mathrm{Arg}(r, j, p_i)$ is $\geq \mathrm{size}(M_{q'}(s_\nu))$. Thus, $\mathrm{Arg}(r, j, p_i)$ is infinite.

(iii) Let $s \in T_\Sigma$ and $u \in V(s)$ such that $\hat{M}_{q_0}(s[u \leftarrow p])$ has a subtree $\langle\!\langle q, p\rangle\!\rangle(\xi_1, \ldots, \xi_n)$ for trees $\xi_1, \ldots, \xi_n$ and $\xi_\mu[\![\mathrm{rhs}]\!]$ has a subtree $\langle q', x_\nu\rangle(\bar{t}_1, \ldots, \bar{t}_l)$ for some $q' \in Q^{(l)}$, $l \geq 0$, $\nu \in [k] - \{i\}$, and trees $\bar{t}_1, \ldots, \bar{t}_l$. It follows from Lemma 2.6 ($S_1^{\langle q', x_\nu\rangle} = 0$) that ξ_μ contains some $\langle\!\langle q'', p\rangle\!\rangle$, $q'' \in Q$, such that $\mathrm{rhs}_M(q'', \sigma, \langle p_1, \ldots, p_k\rangle)$ contains $\langle q', x_\nu\rangle$. Since $\langle\!\langle q'', p\rangle\!\rangle$ is reachable (because ξ_μ is a subtree of $\hat{M}_{q_0}(s[u \leftarrow p])$), $\langle\!\langle q', p_\nu\rangle\!\rangle$ is reachable by the same argument as used above $(*)$. Thus, $\mathrm{Out}(q', p_\nu)$ is infinite. Let $j \in [m]$ such that y_μ occurs in t_j. Then, by Lemma 2.1, t'_j has a subtree $M_{q'}(s_\nu)[y_\eta \leftarrow \bar{t}_\eta\Psi_{s_1,\ldots,s_k}[\![i]\!] \mid \eta \in [l]]$, the size of which is $\geq \mathrm{size}(M_{q'}(s_\nu))$. Letting s_ν range over L_{p_ν} in $(*)$ this implies, analogous to case (ii), that $\mathrm{Arg}(r, j, p_i)$ is infinite. □

We are now ready to prove that properness (i.e., i-properness, p-properness, and productivity) is a normal form for $\mathrm{MTT^R}$s.

THEOREM 5.9. *For every MTT^R M there is (effectively) a proper MTT^R $\mathrm{prop}(M)$ equivalent to M. If M is a T^R, then so is $\mathrm{prop}(M)$.*

Proof. Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$. By Lemma 5.4 we may assume that M is productive and i-proper. Let $q \in Q^{(n)}$ and $p \in P$. The idea of constructing $\mathrm{prop}(M)$ is to delete all parameters y_j of q for which $\mathrm{Arg}(q, j, p)$ is finite and to keep the parameters $y_{j_1}, \ldots, y_{j_m}$ of q for which $\mathrm{Arg}(q, j_\nu, p)$ is infinite. The information on the actual parameter tree which has to be substituted for y_j is stored in the states of $\mathrm{prop}(M)$. More precisely, a state of $\mathrm{prop}(M)$ will be of the form (q, φ), where φ is a mapping which associates with j_ν the new parameter y_ν, and with j a tree ξ_j in the finite set $\mathrm{Arg}(q, j, p)$.

Let us first define an auxiliary notion. For every $q \in Q^{(n)}$, $n \geq 0$, and $p \in P$, let $\Phi_{q,p}$ be the (finite) set of all mappings φ from $[n]$ to $T_{\langle\langle Q,\{p\}\rangle\rangle \cup \Delta} \cup Y$ such that there are $s \in T_\Sigma$, $u \in V(s)$, and $\xi_1, \ldots, \xi_n \in T_{\langle\langle Q,\{p\}\rangle\rangle \cup \Delta}$: $\hat{M}_{q_0}(s[u \leftarrow p])$ has a subtree $\langle\langle q, p\rangle\rangle(\xi_1, \ldots, \xi_n)$ and $F_{q,p}(\varphi, \xi_1, \ldots, \xi_n)$. The predicate $F_{q,p}(\varphi, \xi_1, \ldots, \xi_n)$ holds if for all $j \in [n]$ the following holds: if $j = j_\eta$ for an $\eta \in [m]$, then $\varphi(j) = y_\eta$, and otherwise $\varphi(j) = \xi_j$, where $\{j_1, \ldots, j_m\} = \{j \in [n] \mid \mathrm{Arg}(q, j, p) \text{ is infinite}\}$ and $j_1 < \cdots < j_m$.

By the definition of Arg, $\varphi(j) \notin Y$ implies $\varphi(j) \in \mathrm{Arg}(q, j, p)$. Note that $\Phi_{q,p}$ is finite because $\varphi(j) \in Y_m \cup K_j$ with $K_j = \mathrm{Arg}(q, j, p)$ for finite $\mathrm{Arg}(q, j, p)$, and $K_j = \varnothing$ otherwise. Therefore, $\Phi_{q,p}$ can be obtained effectively by checking, for the (finitely many) mappings $\varphi : [n] \to K$, whether or not $\varphi \in \Phi_{q,p}$ (where $K = Y_m \cup \bigcup_{j \in [n]} K_j$ can be constructed by Lemma 5.7). This is decidable because, apart from the requirement that $\varphi(j_\eta) = y_\eta$ for all $\eta \in [m]$ (which is decidable by Lemma 5.7), φ is in $\Phi_{q,p}$ if and only if $\tau_{\hat{M}}^{-1}(L) \cap S$ is nonempty, where $S = \{s[u \leftarrow p] \mid s \in T_\Sigma, u \in V(s)\}$ and L consists of all trees in $T_{\langle\langle Q,\{p\}\rangle\rangle \cup \Delta}$ which have a subtree $\langle\langle q, p\rangle\rangle(\xi_1, \ldots, \xi_n)$ with $\xi_j = \varphi(j)$ for all $j \notin \varphi^{-1}(Y)$. Clearly, L is regular and hence, by Lemma 3.7, $\tau_{\hat{M}}^{-1}(L)$ is regular. Since S is regular, so is $\tau_{\hat{M}}^{-1}(L) \cap S$, which implies that its emptiness is decidable.

We first construct the $\mathrm{MTT^R}$ $\pi(M)$ by deleting, in the right-hand side of a rule (with look-ahead $\langle p_1, \ldots, p_k\rangle$), all parameters y_j of $\langle r, x_i\rangle$ for which $\mathrm{Arg}(r, j, p_i)$ is finite and replacing them by the appropriate tree in $\mathrm{Arg}(r, j, p_i)$. This tree is coded in the states of $\pi(M)$. Due to the new states of $\pi(M)$, a parameter y_{j_ν} of r with $\mathrm{Arg}(r, j_\nu, p_i)$ infinite might correspond in $\pi(M)$ to the parameter y_ν of a state (r, φ) with finite $\mathrm{Arg}((r, \varphi), \nu, p_i)$. For this reason we have to iterate the application of π (as in the construction in the proof of Lemma 5.4) until the ranks of the states do not change anymore. This results in the desired $\mathrm{MTT^R}$ $\mathrm{prop}(M)$.

Define $\pi(M) = (Q', P, \Sigma, \Delta, (q_0, \varnothing), R', h)$ with $Q' = \{(q, \varphi)^{(m)} \mid q \in Q, \exists p \in P : \varphi \in \Phi_{q,p}, m = |\varphi^{-1}(Y)|\}$. For every $(q, \varphi) \in Q'^{(m)}$, $\sigma \in \Sigma^{(k)}$, $q \in Q^{(n)}$, $m, n, k \geq 0$, and $p, p_1, \ldots, p_k \in P$ with $p = h_\sigma(p_1, \ldots, p_k)$, let the rule

$$\langle (q, \varphi), \sigma(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to \zeta \quad \langle p_1, \ldots, p_k\rangle$$

be in R' such that if $\varphi \notin \Phi_{q,p}$, then ζ is an arbitrary ("dummy") tree in $T_\Delta(Y_m) - Y$ with $\#_{y_j}(\zeta) = 1$ for every $j \in [m]$, and if $\varphi \in \Phi_{q,p}$, then $\zeta = \mathrm{repl}(\mathrm{rhs}(\rho)\Pi)$, where ρ is the $(q, \sigma, \langle p_1, \ldots, p_k\rangle)$-rule of M, Π denotes the substitution

$$[y_j \leftarrow \varphi(j)[\![\mathrm{rhs}]\!] \mid j \in [n]] \quad \text{with} \quad [\![\mathrm{rhs}]\!] = [\![\langle\langle r, p\rangle\rangle \leftarrow \mathrm{rhs}_M(r, \sigma, \langle p_1, \ldots, p_k\rangle) \mid r \in Q]\!],$$

and for every subtree $t \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$ of $\mathrm{rhs}(\rho)\Pi$ the tree $\mathrm{repl}(t)$ is recursively defined as follows:

- for $t \in Y_m$, $\mathrm{repl}(t) = t$,
- for $t = \delta(t_1, \ldots, t_l)$ with $\delta \in \Delta^{(l)}$, $l \geq 0$, and $t_1, \ldots, t_l \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$, $\mathrm{repl}(t) = \delta(\mathrm{repl}(t_1), \ldots, \mathrm{repl}(t_l))$, and
- for $t = \langle q', x_i\rangle(t_1, \ldots, t_l)$, $\langle q', x_i\rangle \in \langle Q, X_k\rangle^{(l)}$, $l \geq 0$, and $t_1, \ldots, t_l \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$,

$$\mathrm{repl}(t) = \langle (q', \varphi'), x_i\rangle(\mathrm{repl}(t_{j_1}), \ldots, \mathrm{repl}(t_{j_\mu})),$$

where $\{j_1, \ldots, j_\mu\} = \{j \in [l] \mid \mathrm{Arg}(q', j, p_i) \text{ is infinite}\}$, $j_1 < \cdots < j_\mu$, and for $j \in [l]$,

$$\varphi'(j) = \begin{cases} y_\eta & \text{if } j = j_\eta \text{ for an } \eta \in [\mu], \\ t_j[\![i]\!] & \text{otherwise} \end{cases}$$

with $[\![i]\!] = [\![\langle r, x_i\rangle \leftarrow \langle\!\langle r, p_i\rangle\!\rangle \mid r \in Q]\!]$.

This ends the construction of $\pi(M)$.

Well-definedness of $\pi(M)$. To prove that $\pi(M)$ is well defined, we have to show that $\mathrm{repl}(\mathrm{rhs}(\rho)\Pi)$ is in $T_{\langle Q', X_k\rangle \cup \Delta}(Y_m)$. Since $\mathrm{rhs}(\rho) \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_n)$ and $\varphi(Y_n) \subseteq Y_m \cup T_{\langle\!\langle Q, \{p\}\rangle\!\rangle \cup \Delta}$ (because $\varphi \in \Phi_{q,p}$), it follows that $\mathrm{rhs}(\rho)\Pi \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$. To prove that $\mathrm{repl}(\mathrm{rhs}(\rho)\Pi) \in T_{\langle Q', X_k\rangle \cup \Delta}(Y_m)$ we must show that, in the definition of repl, if $\langle q', x_i\rangle(t_1, \ldots, t_l)$ is a subtree of $\mathrm{rhs}(\rho)\Pi$, then $(q', \varphi') \in Q'$; i.e., there is a p' such that $\varphi' \in \Phi_{q',p'}$.

We will show that $\varphi' \in \Phi_{q',p_i}$, i.e., that there are $s' \in T_\Sigma$, $u' \in V(s')$, and $\xi'_1, \ldots, \xi'_l \in T_{\langle\!\langle Q, \{p_i\}\rangle\!\rangle \cup \Delta}$ such that $\langle\!\langle q', p_i\rangle\!\rangle(\xi'_1, \ldots, \xi'_l)$ is a subtree of $\hat{M}_{q_0}(s'[u' \leftarrow p_i])$ and $F_{q',p_i}(\varphi', \xi'_1, \ldots, \xi'_l)$. Since $\varphi \in \Phi_{q,p}$, there are $s \in T_\Sigma$, $u \in V(s)$, and $\xi_1, \ldots, \xi_n \in T_{\langle\!\langle Q, \{p\}\rangle\!\rangle \cup \Delta}$ such that $\langle\!\langle q, p\rangle\!\rangle(\xi_1, \ldots, \xi_n)$ is a subtree of $\hat{M}_{q_0}(s[u \leftarrow p])$ and $F_{q,p}(\varphi, \xi_1, \ldots, \xi_n)$. Note in particular that $\langle\!\langle q, p\rangle\!\rangle$ is reachable. Take $s' = s[u \leftarrow \sigma(s_1, \ldots, s_k)]$ with $s_\nu \in L_{p_\nu}$ for all $\nu \in [k]$, and take $u' = ui$. The s_ν exist, because M is i-proper (point (iii)). By Lemma 4.3, $\hat{M}_{q_0}(s'[u' \leftarrow p_i])$ equals $\hat{M}_{q_0}(s[u \leftarrow p])[\![\mathrm{rhs}]\!][..][\![i]\!]$, where $[\![..]\!]$ denotes $[\![\langle r, x_\nu\rangle \leftarrow M_r(s_\nu) \mid \langle r, x_\nu\rangle \in \langle Q, X_k - \{x_i\}\rangle]\!]$, and $[\![\mathrm{rhs}]\!]$ and $[\![i]\!]$ are as in the definition of $\pi(M)$. Since $\langle\!\langle q, p\rangle\!\rangle(\xi_1, \ldots, \xi_n)$ is a subtree of $\hat{M}_{q_0}(s[u \leftarrow p])$ it follows, by Lemma 2.1 and the fact that $[\![..]\!]$ is nondeleting by Lemma 3.11(1), that $\hat{M}_{q_0}(s'[u' \leftarrow p_i])$ has a subtree $\mathrm{rhs}(\rho)\Pi'[\![..]\!][\![i]\!]$, where $\Pi' = [y_\mu \leftarrow \xi_\mu[\![\mathrm{rhs}]\!] \mid \mu \in [n]]$.

Consider the following two cases: (i) there are $t'_1, \ldots, t'_l \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_n)$ such that $\langle q', x_i\rangle(t'_1, \ldots, t'_l)$ is a subtree of $\mathrm{rhs}(\rho)$ and $t'_j\Pi = t_j$ for all $j \in [l]$, and (ii) $\langle q', x_i\rangle(t_1, \ldots, t_l)$ is a subtree of $\varphi(\mu)[\![\mathrm{rhs}]\!]$ for some $\mu \in [n]$.

(i) Since $\mathrm{rhs}(\rho)$ has a subtree $\langle q', x_i\rangle(t'_1, \ldots, t'_l)$, it follows, by application of $\Pi'[\![..]\!][\![i]\!]$ (and Lemma 2.1), that $\hat{M}_{q_0}(s'[u' \leftarrow p_i])$ has a subtree $\langle\!\langle q', p_i\rangle\!\rangle(\xi'_1, \ldots, \xi'_l)$ with $\xi'_j = t'_j\Pi'[\![..]\!][\![i]\!]$ for every $j \in [l]$. Let $j \in [l]$ such that $\mathrm{Arg}(q', j, p_i)$ is finite. Then by Lemma 5.8(ii) and (iii), both t'_j and all $\xi_\mu[\![\mathrm{rhs}]\!]$ such that y_μ occurs in t'_j do not contain elements of $\langle Q, X_k - \{x_i\}\rangle$. Thus $\xi'_j = t'_j\Pi'[\![..]\!][\![i]\!]$ equals $t'_j\Pi'[\![i]\!]$. By Lemma 5.8(i), t'_j does not contain any $y_\mu \in Y_n$ such that $\mathrm{Arg}(q, \mu, p)$ is infinite. Thus, since $F_{q,p}(\varphi, \xi_1, \ldots, \xi_n)$, $t'_j\Pi'[\![i]\!] = t'_j\Pi[\![i]\!] = t_j[\![i]\!]$. By the definition of φ' this shows that $F_{q',p_i}(\varphi', \xi'_1, \ldots, \xi'_l)$.

(ii) There is an occurrence of y_μ in $\mathrm{rhs}(\rho)$, because M is nondeleting. Since $\varphi(\mu) = \xi_\mu$, by the fact that $F_{q,p}(\varphi, \xi_1, \ldots, \xi_n)$ holds, this means that in $\mathrm{rhs}(\rho)\Pi'[\![..]\!][\![i]\!]$ there is a subtree $\langle\!\langle q', p_i\rangle\!\rangle(\xi'_1, \ldots, \xi'_l)$ with $\xi'_j = t_j[\![..]\!][\![i]\!]$ for $j \in [l]$. Since $\langle q', x_i\rangle(t_1, \ldots, t_l)$ is a subtree of $\xi_\mu[\![\mathrm{rhs}]\!]$, it follows from the definition of second-order tree substitution that ξ_μ has a subtree $\langle\!\langle q'', p\rangle\!\rangle(\zeta_1, \ldots, \zeta_\lambda)$ and the right-hand side of the $(q'', \sigma, \langle p_1, \ldots, p_k\rangle)$-rule ρ'' has a subtree $\langle q', x_i\rangle(t'_1, \ldots, t'_l)$ such that $t_j = t'_j[y_\nu \leftarrow \zeta_\nu[\![\mathrm{rhs}]\!] \mid \nu \in [\lambda]]$ for

every $j \in [l]$. Note that $\langle\langle q'', p\rangle\rangle$ is reachable because it occurs in ξ_μ. Now let $j \in [l]$ such that $\mathrm{Arg}(q', j, p_i)$ is finite. Then, as in case (i), by Lemma 5.8(ii) and (iii) applied to ρ'', both t'_j and all $\zeta_\nu[\![\mathrm{rhs}]\!]$ such that y_ν occurs in t'_j do not contain elements of $\langle Q, X_k - \{x_i\}\rangle$. Hence t_j does not contain elements of $\langle Q, X_k - \{x_i\}\rangle$ and thus $\xi'_j = t_j[\![..]\!][\![i]\!] = t_j[\![i]\!]$. By the definition of φ' this shows that $F_{q',p_i}(\varphi', \xi'_1, \ldots, \xi'_l)$.

Equivalence of $\pi(M)$ and M. We now prove that $\pi(M)$ realizes the same translation as M. This follows from Claim 1 for $(q, \varphi) = (q_0, \varnothing)$.

Claim 1. Let $s \in T_\Sigma$, $q \in Q^{(n)}$, $n \geq 0$, and $p = h(s)$. For every $\varphi \in \Phi_{q,p}$, $\pi(M)_{(q,\varphi)}(s) = M_q(s)\Pi'$, where $\Pi' = [y_j \leftarrow \varphi(j)[\![\langle\langle r, p\rangle\rangle \leftarrow M_r(s) \mid r \in Q]\!] \mid j \in [n]]$.

This claim is proved by induction on the structure of s. Let the induction hypothesis be denoted by IH1. Let $s = \sigma(s_1, \ldots, s_k)$ with $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $s_1, \ldots, s_k \in T_\Sigma$. For $i \in [k]$ let $p_i = h(s_i)$ and let $m = \mathrm{rank}_{Q'}((q, \varphi))$.

By Lemma 3.5, $\pi(M)_{(q,\varphi)}(\sigma(s_1, \ldots, s_k)) = \mathrm{rhs}_{\pi(M)}((q, \varphi), \sigma, \langle p_1, \ldots, p_k\rangle)[\![\_]\!]$, where $[\![\_]\!] = [\![\langle r, x_i\rangle \leftarrow \pi(M)_r(s_i) \mid \langle r, x_i\rangle \in \langle Q', X_k\rangle]\!]$. By the definition of the right-hand sides of the rules of $\pi(M)$ we get $\mathrm{repl}(\mathrm{rhs}(\rho)\Pi)[\![\_]\!]$, where repl, ρ, and Π are as in the definition of the rules of $\pi(M)$.

For $t = \mathrm{rhs}(\rho)\Pi$ it follows from Claim 2 that $\mathrm{repl}(\mathrm{rhs}(\rho)\Pi)[\![\_]\!] = \mathrm{rhs}(\rho)\Pi[\![\ldots]\!]$, where $[\![\ldots]\!] = [\![\langle r, x_i\rangle \leftarrow M_r(s_i) \mid \langle r, x_i\rangle \in \langle Q, X_k\rangle]\!]$. If we apply $[\![\ldots]\!]$ to $\mathrm{rhs}(\rho)\Pi$ and use Lemma 3.5 for M, then we get $M_q(s)\Pi'$, which proves Claim 1.

Claim 2. Let $t \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$ be a subtree of $\mathrm{rhs}(\rho)\Pi$. Then $\mathrm{repl}(t)[\![\_]\!] = t[\![\ldots]\!]$.

This claim is proved by induction on the structure of t. The induction hypothesis is denoted by IH2.

If $t \in Y_m$, then $\mathrm{repl}(t)[\![\_]\!] = t[\![\_]\!] = t = t[\![\ldots]\!]$. If $t = \delta(t_1, \ldots, t_l)$ with $\delta \in \Delta^{(l)}$, $l \geq 0$, and $t_1, \ldots, t_l \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$, then $\mathrm{repl}(\delta(t_1, \ldots, t_l))[\![\_]\!]$ equals $\delta(\mathrm{repl}(t_1)[\![\_]\!], \ldots, \mathrm{repl}(t_l)[\![\_]\!])$. By IH2 this equals $\delta(t_1[\![\ldots]\!], \ldots, t_l[\![\ldots]\!]) = t[\![\ldots]\!]$.

If $t = \langle q', x_i\rangle(t_1, \ldots, t_l)$ with $\langle q', x_i\rangle \in \langle Q, X_k\rangle^{(l)}$, $l \geq 0$, and $t_1, \ldots, t_l \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$, then $\mathrm{repl}(t)[\![\_]\!]$ equals $\langle(q', \varphi'), x_i\rangle(\mathrm{repl}(t_{j_1}), \ldots, \mathrm{repl}(t_{j_\mu}))[\![\_]\!]$ with $\{j_1, \ldots, j_\mu\} = \{j \in [l] \mid \mathrm{Arg}(q', j, p_i) \text{ is infinite}\}$ and φ' as in the definition of repl. Applying the substitution $[\![\_]\!]$ we get

$$\pi(M)_{(q',\varphi')}(s_i)[y_\eta \leftarrow \mathrm{repl}(t_{j_\eta})[\![\_]\!] \mid \eta \in [\mu]].$$

Since $\varphi' \in \Phi_{q',p_i}$ (as shown for the well-definedness of $\pi(M)$), we can apply IH1 to $\pi(M)_{(q',\varphi')}(s_i)$ and IH2 to $\mathrm{repl}(t_{j_\eta}[\![\_]\!])$ to get

$$M_{q'}(s_i)\Pi''[y_\eta \leftarrow t_{j_\eta}[\![\ldots]\!] \mid \eta \in [\mu]]$$

with $\Pi'' = [y_j \leftarrow \varphi'(j)[\![.]\!] \mid j \in [l]]$ and $[\![.]\!] = [\![\langle\langle r, p_i\rangle\rangle \leftarrow M_r(s_i) \mid r \in Q]\!]$.

By the definition of φ' we can write this as

$$M_{q'}(s_i)[y_j \leftarrow t_j[\![i]\!][\![.]\!] \mid j \in [l], j \neq j_\eta \text{ for } \eta \in [\mu]] \\ [y_{j_\eta} \leftarrow y_\eta \mid \eta \in [\mu]]\,[y_\eta \leftarrow t_{j_\eta}[\![\ldots]\!] \mid \eta \in [\mu]].$$

Since $\varphi' \in \Phi_{q',p_i}$, t_j is in $T_{\langle Q, \{x_i\}\rangle \cup \Delta}$ for $j \neq j_\eta$. Therefore, in $t_j[\![i]\!][\![.]\!] = t_j[\![\langle r, x_i\rangle \leftarrow M_r(s_i) \mid r \in Q]\!]$ we can extend the substitution to all elements of $\langle Q, X_k\rangle$ to get $t_j[\![\ldots]\!]$. Altogether we get

$$M_{q'}(s_i)[y_j \leftarrow t_j[\![\ldots]\!] \mid j \in [l], j \neq j_\eta \text{ for } \eta \in [\mu]][y_{j_\eta} \leftarrow t_{j_\eta}[\![\ldots]\!] \mid \eta \in [\mu]],$$

which equals $M_{q'}(s_i)[y_j \leftarrow t_j[\![\ldots]\!] \mid j \in [l]] = \langle q', x_i\rangle(t_1, \ldots, t_l)[\![\ldots]\!]$. This ends the proof of Claim 2.

Nondeleting of $\pi(M)$. Consider the $((q,\varphi),\sigma,\langle p_1,\dots,p_k\rangle)$-rule r of $\pi(M)$ and let $\varphi^{-1}(Y_m)=\{j_1,\dots,j_m\}$ with $j_1<\cdots<j_m$. Let $\nu\in[m]$. If r is a dummy rule, then $\#_{y_\nu}(\mathrm{rhs}(r))=1$. Otherwise $\mathrm{rhs}(r)=\mathrm{repl}(\mathrm{rhs}(\rho)\Pi)$, where ρ is the $(q,\sigma,\langle p_1,\dots,p_k\rangle)$-rule of M. Since M is nondeleting, y_{j_ν} occurs in $\mathrm{rhs}(\rho)$. Since $\varphi\in\Phi_{q,p}$, $\varphi(j_\nu)=y_\nu$; this means that the substitution Π replaces y_{j_ν} by y_ν, and hence y_ν occurs in $\mathrm{rhs}(\rho)\Pi$. To show that y_ν occurs in $\mathrm{repl}(\mathrm{rhs}(\rho)\Pi)$, we prove that for $t\in T_{\langle Q,X_k\rangle\cup\Delta}(Y_m)$, if y_ν occurs in t, then it also occurs in $\mathrm{repl}(t)$. The proof is by induction on the structure of t. It is obvious for $t\in Y_m$ and $t=\delta(t_1,\dots,t_l)$. For $t=\langle q',x_i\rangle(t_1,\dots,t_l)$, let $j\in[l]$ such that y_ν occurs in t_j, and let φ' be as in the definition of repl. By induction, y_ν occurs in $\mathrm{repl}(t_j)$. Then y_ν occurs also in $t_j[\![i]\!]$, where $[\![i]\!]$ is as in the definition of repl. This means that $t_j[\![i]\!]\notin T_{\langle\langle Q,\{p_i\}\rangle\rangle\cup\Delta}$ and since $\varphi'\in\Phi_{q',p_i}$, this implies that $\varphi'(j)=y_\eta$ for some $\eta\in[\mu]$ with $j=j'_\eta$, where $\{j'_1,\dots,j'_\mu\}={\varphi'}^{-1}(Y_\mu)$ and $j'_1<\cdots<j'_\mu$. By the definition of repl, $\mathrm{repl}(t_{j'_\eta})=\mathrm{repl}(t_j)$ is a subtree of $\mathrm{repl}(t)$ and therefore y_ν occurs in $\mathrm{repl}(t)$.

Nonerasing of $\pi(M)$. Clearly, from the definition of repl, if $\mathrm{repl}(t)\in Y$, then $t\in Y$. Hence $\mathrm{repl}(\mathrm{rhs}(\rho)\Pi)\in Y$ implies $\mathrm{rhs}(\rho)\Pi\in Y$ and so, obviously, $\mathrm{rhs}(\rho)\in Y$. Thus, since M is nonerasing, so is $\pi(M)$.

I-properness of $\pi(M)$. Since $\pi(M)$ has the same look-ahead automaton as M, point (iii) of i-properness is preserved. It follows from the definition of Π and repl and from i-properness of M that no (q_0,φ) appears in the right-hand side of a rule of $\pi(M)$. Using Lemma 4.3 (and the fact that, in the definition of $\mathrm{repl}(t)$, $\varphi'\in\Phi_{q',p_i}$) it is not difficult to see that if $\langle\langle(q,\varphi),p\rangle\rangle$ is reachable, then $\varphi\in\Phi_{q,p}$ and hence, by the definition of $\Phi_{q,p}$, $\langle\langle q,p\rangle\rangle$ is reachable. Also by Lemma 4.3, if $(q,\varphi)\neq(q_0,\varnothing)$, then (q,φ) appears in the right-hand side of a rule of $\pi(M)$, and so $q\neq q_0$. By Claim 1, $\pi(M)_{(q,\varphi)}(s)=M_q(s)\Pi'$ with $\Pi'=[y_j\leftarrow\varphi(j)[\![\langle\langle r,p\rangle\rangle\leftarrow M_r(s)\mid r\in Q]\!]\mid j\in[n]]$. Since $\mathrm{size}(M_q(s)\Pi')\geq\mathrm{size}(M_q(s))$, $\mathrm{Out}((q,\varphi),p)=\{M_q(s)\Pi'\mid s\in L_p\}$ is infinite if $\{M_q(s)\mid s\in L_p\}=\mathrm{Out}(q,p)$ is infinite, which holds by i-properness of M.

P-properness. By constructing $\pi(M)$ we have kept only those parameter positions j of q for which $\mathrm{Arg}(q,j,p)$ is infinite. But even if $\mathrm{Arg}(q,j,p)$ is infinite, there might be a $\varphi\in\Phi_{q,p}$ for which $\mathrm{Arg}((q,\varphi),j,p)$ is finite. This means that $\pi(M)$ need not be p-proper yet (see Example 5.10), and, as in the case of i-properness, we have to iterate the application of π. For the termination condition of this iteration we only need to consider particular states which are actually used in the derivations of $\pi^k(M)$. Denote the state $(\cdots((q,\varphi_1),\varphi_2)\dots,\varphi_k)$ of $\pi^k(M)$ by $(q,\varphi_1,\dots,\varphi_k)$. The state $(q,\varphi_1,\dots,\varphi_k)$ is *p-uniform* if for each $0\leq i\leq k-1$, $\varphi_{i+1}\in\Phi_{(q,\varphi_1,\dots,\varphi_i),p}$. We iterate the application of π until we obtain the $\mathrm{MTT}^{\mathrm{R}}$ N (with set of states Q_N) such that

$(*)$ for every $p\in P$ and p-uniform state (q,φ) of $M'=\pi(N)$,
$$\mathrm{rank}_{Q'}((q,\varphi))=\mathrm{rank}_{Q_N}(q),$$

where Q' is the set of states of M'.

Let us now show that, indeed, after a finite number of applications of π, $(*)$ holds. For $q\in Q$ and $p\in P$, define the tree $T_{q,p}$ as follows. For $k\geq 0$, the state $(q,\varphi_1,\dots,\varphi_k)$ of $\pi^k(M)$ is a node of $T_{q,p}$ if it is p-uniform and there is a p-uniform state $(q,\varphi_1,\dots,\varphi_k,\dots,\varphi_l)$ of $\pi^l(M)$ with $l>k$ which is of smaller rank than $(q,\varphi_1,\dots,\varphi_k)$. There is an edge in $T_{q,p}$ from every node $(q,\varphi_1,\dots,\varphi_k)$ to every node $(q,\varphi_1,\dots,\varphi_k,\varphi_{k+1})$. Clearly, if $T_{q,p}$ is finite for every $q\in Q$ and $p\in P$, then the iteration of π terminates: Let l be maximal such that $(q,\varphi_1,\dots,\varphi_l)$ is a leaf of $T_{q,p}$ for some $q\in Q$ and $p\in P$. Then the statement in $(*)$ holds for $N=\pi^{l+1}(M)$, because

no p-uniform state $(q, \varphi_1, \ldots, \varphi_l, \varphi_{l+1})$ is a node of $T_{q,p}$ and hence, by the definition of the nodes of $T_{q,p}$, every p-uniform state $(q, \varphi_1, \ldots, \varphi_{l+1}, \varphi_{l+2})$ has the same rank as $(q, \varphi_1, \ldots, \varphi_{l+1})$. To show the finiteness of $T_{q,p}$ it suffices, by König's lemma, to show that every path ρ of $T_{q,p}$ is finite. Assume to the contrary that ρ is infinite. Let $u = (q, \varphi_1, \ldots, \varphi_k)$ be a node of ρ. Then there is a descendant of u on the path ρ, that has lower rank than u. This can be seen as follows: By the definition of the node u, there is a p-uniform state $(q, \varphi_1, \ldots, \varphi_k, \ldots, \varphi_l)$ of $\pi^l(M)$, $l > k$, which has lower rank than u. Now, for each $i \in \{k+1, \ldots, l\}$ such that $v = (q, \varphi_1, \ldots, \varphi_k, \ldots, \varphi_{i-1})$ is on the path ρ, either $v' = (v, \varphi_i) = (q, \varphi_1, \ldots, \varphi_k, \ldots, \varphi_i)$ has the same rank as v and then v' is on the path ρ because $\Phi_{v,p} = \{v'\}$ by the definition of $\Phi_{v,p}$ or v' has a lower rank n than v, and then, by the definition of $\Phi_{v,p}$, each state (v, φ) has rank n, in particular the child of v that is on the path ρ. Since each node u of ρ has a descendant on ρ that has a lower rank than u, there is an infinite sequence of nodes on ρ with strictly decreasing ranks. This contradicts the finiteness of the rank of q.

Before we show that M' is p-proper, we prove a claim about p-uniformity.

Claim 3. Let $k \geq 0$, let q be a state of $\pi^k(M)$, and let $p \in P$.

(i) If $\langle q, x_i \rangle$ appears in the right-hand side of a $(q', \sigma, \langle p_1, \ldots, p_{k'} \rangle)$-rule of $\pi^k(M)$ for some state q' of $\pi^k(M)$, $k' \geq 0$, $i \in [k']$, and $p_1, \ldots, p_{k'} \in P$, then q is p_i-uniform.

(ii) If $\langle\langle q, p \rangle\rangle$ is reachable (by $\pi^k(M)$), then q is p-uniform.

The proof of part (i) of Claim 3 is by induction on k. For $k = 0$, every state is p-uniform for all $p \in P$, and thus the statement holds. Now assume the statement holds for $\pi^k(M)$. If $\langle (q, \varphi), x_i \rangle$ appears in the right-hand side ζ of the $((q', \varphi'), \sigma, \langle p_1, \ldots, p_{k'} \rangle)$-rule of $\pi(\pi^k(M))$, then, by the definition of the rules of $\pi(\pi^k(M))$, ζ is of the form $\mathrm{repl}(\mathrm{rhs}(\rho)\Pi)$, where ρ is the $(q', \sigma, \langle p_1, \ldots, p_{k'} \rangle)$-rule of $\pi^k(M)$. Thus, by the definition of repl and Π, $\langle q, x_i \rangle$ occurs in $\mathrm{rhs}(\rho)$, which means, by induction, that q is p_i-uniform. In the proof of well-definedness of $\pi(M)$ it is shown that $\varphi \in \Phi_{q,p_i}$, and hence also (q, φ) is p_i-uniform. This proves part (i) of the claim. To prove part (ii), we may assume that $q \neq r_0$, the initial state of $\pi^k(M)$; in fact, $r_0 = (q_0, \varnothing, \ldots, \varnothing)$ is p-uniform for every p. If $\langle\langle q, p \rangle\rangle$ is reachable (by $\pi^k(M)$), then, by definition, it appears in $\widehat{\pi^k(M)}_{r_0}(s[u \leftarrow p])$ for some tree s and node u of s, where $\widehat{\pi^k(M)}$ denotes the extension of $\pi^k(M)$. Since $q \neq r_0$, u must be of the form $u'j$ with $u' \in \mathbb{N}^*$ and $j \geq 1$. Hence, by Lemma 4.3, $\langle q, x_j \rangle$ must occur in the right-hand side of some rule of $\pi^k(M)$ with look-ahead $\langle p_1, \ldots, p_l \rangle$, $l \geq 1$, and $p_j = p$. By part (i) of the claim this implies that q is p-uniform. This concludes the proof of Claim 3.

Let us now prove (i) of p-properness for N. Let $\langle\langle q, p \rangle\rangle$ be reachable (by N). By Claim 3(ii), q is p-uniform. Since $\langle\langle q, p \rangle\rangle$ is reachable, the set $\Phi_{q,p}$ must, by definition, contain some element φ. Then (q, φ) is p-uniform, and it follows from $(*)$ that $n = |\varphi^{-1}(Y)|$ and thus $\{j \in [n] \mid \mathrm{Arg}(q, j, p) \text{ is infinite}\} = \{1, \ldots, n\}$. Thus (i) of p-properness holds for N. Now consider M'. Note that, by the previous argument, if (q, φ) is a p-uniform state of M', then $\varphi = \varphi_n$, where $q \in Q_N^{(n)}$ and $\varphi_n(j) = y_j$ for every $j \in [n]$. Clearly, (i) of p-properness also holds for M'. Formally this can be shown by proving that $\mathrm{Arg}((q, \varphi_n), j, p) = \mathrm{Arg}(q, j, p)[\![\mathrm{rel}]\!]$, where $[\![\mathrm{rel}]\!]$ denotes the relabeling $[\![\langle\langle q', p \rangle\rangle \leftarrow \langle\langle (q', \varphi_{n'}), p \rangle\rangle \mid q' \in Q_N^{(n')}, n' \geq 0]\!]$. This follows from Claim 4 (for q equal to the initial state of N and φ equal to $\varnothing$).

Claim 4. Let $s \in T_\Sigma$, $u \in V(s)$, and $p \in P$, and let (q, φ) be an $h(s[u \leftarrow p])$-uniform state of M'. Then

$$\hat{M}'_{(q,\varphi)}(s[u \leftarrow p]) = \hat{N}_q(s[u \leftarrow p])[\![\mathrm{rel}]\!].$$

The proof is by induction on the structure of s. Let $s = \sigma(s_1, \ldots, s_k)$ with $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $s_1, \ldots, s_k \in T_\Sigma$. For $u = \varepsilon$ we get $\hat{M}'_{(q,\varphi)}(s[u \leftarrow p]) = \langle\langle (q, \varphi), p\rangle\rangle$. Since $\varphi = \varphi_n$, where n is the rank of q, $\langle\langle (q, \varphi), p\rangle\rangle = \langle\langle q, p\rangle\rangle[\![\mathrm{rel}]\!] = \hat{N}_q(s[u \leftarrow p])[\![\mathrm{rel}]\!]$. For $u = ju'$ with $j \geq 1$ and $u' \in \mathbb{N}^*$, $s[u \leftarrow p] = \sigma(\tilde{s}_1, \ldots, \tilde{s}_k)$ with $\tilde{s}_j = s_j[u' \leftarrow p]$ and $\tilde{s}_i = s_i$ for $i \in [k] - \{j\}$. By Lemma 3.5 and the definition of the right-hand sides of M', $\hat{M}'_{(q,\varphi)}(s[u \leftarrow p]) = \mathrm{repl}(\mathrm{rhs}(\rho)\Pi)[\![\_]\!]$, where ρ is the $(q, \sigma, \langle h(\tilde{s}_1), \ldots, h(\tilde{s}_k)\rangle)$-rule of N and $[\![\_]\!] = [\![\langle (q', \varphi'), x_i\rangle \leftarrow \hat{M}'_{(q',\varphi')}(\tilde{s}_i) \mid \langle (q', \varphi'), x_i\rangle \in \langle Q', X_k\rangle]\!]$. By Claim 3(i), if $\langle (q', \varphi'), x_i\rangle$ occurs in $\mathrm{repl}(\mathrm{rhs}(\rho)\Pi)$, then (q', φ') is $h(\tilde{s}_i)$-uniform and, by the argument given above Claim 4, $\varphi' = \varphi_{n'}$, where $q' \in Q_N^{(n')}$. Clearly, $\mathrm{repl}(\mathrm{rhs}(\rho)\Pi)$ equals $\mathrm{rhs}(\rho)[\![\ ]\!]$ with $[\![\ ]\!] = [\![\langle q', x_i\rangle \leftarrow \langle (q', \varphi_{n'}), x_i\rangle \mid \langle q', x_i\rangle \in \langle Q_N, X_k\rangle^{(n')}, n' \geq 0]\!]$. Furthermore, we can restrict the substitution $[\![\_]\!]$ to those $\langle (q', \varphi'), x_i\rangle$ which occur in $\mathrm{repl}(\mathrm{rhs}(\rho)\Pi)$ and then apply the induction hypothesis to $\tilde{s}_j = s_j[u \leftarrow p]$. If we combine the resulting substitution with $[\![\ ]\!]$ and apply Claim 1 to $\tilde{s}_i = s_i$ for $i \in [k]-\{j\}$ (where Π' is the identity), then we get $\mathrm{rhs}(\rho)[\![\langle q', x_i\rangle \leftarrow \hat{N}_{q'}(\tilde{s}_i)[\![\mathrm{rel}]\!] \mid \langle q', x_i\rangle \in \langle Q_N, X_k\rangle$ occurs in $\mathrm{rhs}(\rho)]\!] = \mathrm{rhs}(\rho)[\![\langle q', x_i\rangle \leftarrow \hat{N}_{q'}(\tilde{s}_i)[\![\mathrm{rel}]\!] \mid \langle q', x_i\rangle \in \langle Q_N, X_k\rangle]\!]$, which equals $\hat{N}_q(s[u \leftarrow p])[\![\mathrm{rel}]\!]$. This proves Claim 4.

To show (ii) of p-properness of M', note that if $\varphi \in \Phi_{q,p}$, then $\langle\langle q, p\rangle\rangle$ is reachable (by N), and hence, by Claim 3(ii), q is p-uniform; then also (q, φ) is p-uniform, $\varphi = \varphi_n$, and, by Claim 4, $\langle\langle (q, \varphi), p\rangle\rangle$ is reachable (by M'). Thus, if $\langle\langle (q, \varphi), p\rangle\rangle$ is not reachable, then $\varphi \notin \Phi_{q,p}$. This implies a dummy right-hand side for all $((q, \varphi), \sigma, \langle p_1, \ldots, p_k\rangle)$-rules with $h_\sigma(p_1, \ldots, p_k) = p$ and therefore $\#_{y_j}(M'_{(q,\varphi)}(s)) = 1$ for all $s \in L_p$. This proves (ii) of p-properness and concludes the proof of properness of M'. Hence, the lemma holds for $\mathrm{prop}(M) = M'$. □

The following example illustrates the construction of a proper $\mathrm{MTT}^{\mathrm{R}}$ as given in the proof of Theorem 5.9.

Example 5.10. Let $M = (Q, \{p\}, \Sigma, \Delta, q_0, R, h)$ be the MTT with $Q = \{q_0^{(0)}, q^{(2)}\}$, $\Sigma = \{a^{(1)}, b^{(1)}, e^{(0)}\}$, $\Delta = \{\sigma^{(3)}, \gamma^{(1)}, a^{(0)}, b^{(0)}, e^{(0)}\}$, and R consisting of the following rules (where the only look-ahead $\langle p\rangle$ is omitted, as usual):

$$\begin{array}{llll}
\langle q_0, a(x_1)\rangle & \to & \langle q, x_1\rangle(a, a), & \langle q, a(x_1)\rangle(y_1, y_2) \to \sigma(y_1, y_2, \langle q, x_1\rangle(a, a)),\\
\langle q_0, b(x_1)\rangle & \to & \langle q, x_1\rangle(b, b), & \langle q, b(x_1)\rangle(y_1, y_2) \to \sigma(y_1, y_2, \langle q, x_1\rangle(b, \gamma(y_2))),\\
\langle q_0, e\rangle & \to & e, & \langle q, e\rangle(y_1, y_2) \to \sigma(y_1, y_2, e).
\end{array}$$

Note that M is productive and i-proper. Let us now construct the MTT $\pi(M)$ as defined in the proof of Theorem 5.9. Clearly, $\mathrm{Arg}(q, 1, p) = \{a, b\}$ and $\mathrm{Arg}(q, 2, p) = \{\gamma^n(c) \mid n \geq 0, c \in \{a, b\}\}$. Thus, $\Phi_{q,p}$ consists of the two mappings φ_a and φ_b with $\varphi_a(1) = a$, $\varphi_a(2) = y_1$, $\varphi_b(1) = b$, and $\varphi_b(2) = y_1$. Therefore the states of $M_1 = \pi(M)$ are $(q_0, \varnothing)^{(0)}, (q, \varphi_a)^{(1)}, (q, \varphi_b)^{(1)}$, abbreviated by q_0, q_a, q_b, respectively. For every $c \in \{a, b\}$, M_1 has the following rules.

$$\begin{array}{llll}
\langle q_0, a(x_1)\rangle & \to & \langle q_a, x_1\rangle(a), & \langle q_c, a(x_1)\rangle(y_1) \to \sigma(c, y_1, \langle q_a, x_1\rangle(a)),\\
\langle q_0, b(x_1)\rangle & \to & \langle q_b, x_1\rangle(b), & \langle q_c, b(x_1)\rangle(y_1) \to \sigma(c, y_1, \langle q_b, x_1\rangle(\gamma(y_1))),\\
\langle q_0, e\rangle & \to & e, & \langle q_c, e\rangle(y_1) \to \sigma(c, y_1, e).
\end{array}$$

Now for M_1, $\mathrm{Arg}(q_a, 1, p) = \{a\}$ and $\mathrm{Arg}(q_b, 1, p) = \{\gamma^n(c) \mid n \geq 0, c \in \{a, b\}\}$. Since $\langle\langle q_a, p\rangle\rangle$ is reachable this means that M_1 is not p-proper.

Following the proof of Theorem 5.9, we have to construct the MTT $N = \pi(M_1)$, because $\mathrm{rank}_{Q'}((q, \varphi_a)) < \mathrm{rank}_Q(q)$. Clearly, $\Phi_{q_a,p} = \{\varphi'_a\}$ with $\varphi'_a(1) = a$, and $\Phi_{q_b,p} = \{\varphi_1\}$ with $\varphi_1(1) = y_1$. Thus, the states of N are $(q_0, \varnothing)^{(0)}, (q_a, \varphi'_a)^{(0)}$, and

$(q_b, \varphi_1)^{(1)}$, abbreviated by q_0, q_a, and q_b, respectively. The rules of N are as follows:

$$\begin{array}{lcl}
\langle q_0, a(x_1)\rangle & \to & \langle q_a, x_1\rangle, \\
\langle q_0, b(x_1)\rangle & \to & \langle q_b, x_1\rangle(b), \\
\langle q_0, e\rangle & \to & e, \\
\langle q_a, a(x_1)\rangle & \to & \sigma(a, a, \langle q_a, x_1\rangle), \\
\langle q_b, a(x_1)\rangle(y_1) & \to & \sigma(b, y_1, \langle q_a, x_1\rangle), \\
\langle q_a, b(x_1)\rangle & \to & \sigma(a, a, \langle q_b, x_1\rangle(\gamma(a))), \\
\langle q_b, b(x_1)\rangle(y_1) & \to & \sigma(b, y_1, \langle q_b, x_1\rangle(\gamma(y_1))), \\
\langle q_a, e\rangle & \to & \sigma(a, a, e), \\
\langle q_b, e\rangle(y_1) & \to & \sigma(b, y_1, e).
\end{array}$$

The MTT N is p-proper because $\mathrm{Arg}(q_b, 1, p) = \{\gamma^n(c) \mid n \geq 0, c \in \{a, b\}\}$ (and all elements of $\langle\langle Q_N, \{p\}\rangle\rangle$ are reachable). It is easy to see that N is equivalent to M.

6. From linear size increase to finite copying. In this section we prove that if a proper $\mathrm{MTT}^{\mathrm{R}}$ M is lsi, then it is finite copying (i.e., both fci and fcp; see section 4.1). The proof is split up into the following three stages, using fnest (see section 4.2) as an intermediate notion:

(I) If M is lsi, then it is fnest.
(II) If M is lsi and fnest, then it is fcp.
(III) If M is lsi, fnest, and fcp, then it is fci.

We first prove (II), then (III), and finally (I). The reason for this order is that the proof of (I) will use results that are proved in (III). The idea in each stage is roughly as follows: First, it is proved that if M's copying is *not* bounded (i.e., M is not fcp, not fci, and not fnest for (II), (III), and (I), respectively), then we can find an input tree in which some part s can be pumped, i.e., repeated; each repetition of s will produce a copy of a certain parameter (for (II)) or of a certain state (for (III) and (I)). Second, it is shown that this repetition gives a size increase that is not linearly bounded (by any c); in this part the properness of M is used: it is shown that for any c we can pick a sufficiently large output tree t, a copy of which is generated with each repetition of s, and a sufficiently large i such that after i repetitions of s the size of the corresponding output tree is larger than c times the size of the input tree.

6.1. From lsi and fnest to fcp (II). We now present (in Lemma 6.2) a pumping lemma for non-fcp $\mathrm{MTT}^{\mathrm{R}}_{\mathrm{fnest}}$s, which allows us to prove (in Theorem 6.3) that if a proper $\mathrm{MTT}^{\mathrm{R}}_{\mathrm{fnest}}$ is lsi, then it is fcp.

First, for an $\mathrm{MTT}^{\mathrm{R}}$ M, consider the number k of occurrences of y_ν in $\hat{M}_r(t[u \leftarrow p])$ with $p = h(t/u)$. Clearly, if $\hat{M}_r(t[u \leftarrow p])$ has a subtree $\langle\langle r_1, p\rangle\rangle(\xi_1, \ldots, \xi_{m_1})$ such that y_ν occurs in ξ_{ν_1} for some $\nu_1 \in [m_1]$, then, assuming that M is nondeleting, the number of y_ν's in $M_r(t)$ must be at least $k-1$ plus the number of y_{ν_1}'s in $M_{r_1}(t/u)$. This is proved in the next lemma, in such a way that the idea can be iterated.

LEMMA 6.1. *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be a nondeleting MTT^R. For $r_0 \in Q^{(m_0)}$, $r_1 \in Q^{(m_1)}$, $\nu_0 \in [m_0]$, $\nu_1 \in [m_1]$, $t_0 \in T_\Sigma$, $u_1 \in V(t_0)$, and $k \in \mathbb{N}$, let $\mathcal{P}(r_0, \nu_0, t_0, r_1, \nu_1, u_1, k)$ be the following statement, with p_1 denoting $h(t_0/u_1)$:*

$$\#_{y_{\nu_0}}(\hat{M}_{r_0}(t_0[u_1 \leftarrow p_1])) \geq k \text{ and } \hat{M}_{r_0}(t_0[u_1 \leftarrow p_1]) \text{ has a subtree } \langle\langle r_1, p_1\rangle\rangle(\xi_1, \ldots, \xi_{m_1}) \text{ for certain } \xi_1, \ldots, \xi_{m_1} \text{ such that } \#_{y_{\nu_0}}(\xi_{\nu_1}) \geq 1.$$

Let $r_2 \in Q^{(m_2)}$, $\nu_2 \in [m_2]$, $u_2 \in V(t_0/u_1)$, and $l \in \mathbb{N}$. If $\mathcal{P}(r_0, \nu_0, t_0, r_1, \nu_1, u_1, k)$ and $\mathcal{P}(r_1, \nu_1, t_0/u_1, r_2, \nu_2, u_2, l)$, then $\mathcal{P}(r_0, \nu_0, t_0, r_2, \nu_2, u_1u_2, k+l-1)$.

Proof. Note that $t_0/u_1u_2 = (t_0/u_1)/u_2$. Let $t_1 = t_0/u_1$, $p_1 = h(t_1)$, and $p_2 = h(t_0/u_1u_2) = h(t_1/u_2)$. By Lemma 4.2, $\hat{M}_{r_0}(t_0[u_1u_2 \leftarrow p_2])$ equals $t[\![\ldots]\!]$ with $t = \hat{M}_{r_0}(t_0[u_1 \leftarrow p_1])$ and $[\![\ldots]\!] = [\![\langle\!\langle q', p_1\rangle\!\rangle \leftarrow \hat{M}_{q'}(t_1[u_2 \leftarrow p_2]) \mid q' \in Q]\!]$. We use Lemma 2.6 to compute the number of occurrences of y_{ν_0}'s in this tree. By the first assumption, t has at least k leaves $u \in V_{y_{\nu_0}}(t)$, and it has a subtree $\langle\!\langle r_1, p_1\rangle\!\rangle(\xi_1, \ldots, \xi_{m_1})$ with $\#_{y_{\nu_0}}(\xi_{\nu_1}) \geq 1$. Thus, t has a leaf $u \in V_{y_{\nu_0}}(t)$ such that $\prod F_{t,u}^{[\![\ldots]\!]} \geq \#_{y_{\nu_1}}(\hat{M}_{r_1}(t_1[u_2 \leftarrow p_2]))$, which is $\geq l$ by the second assumption. Hence, $S_1^{y_{\nu_0}} + S_2^{y_{\nu_0}}$ of Lemma 2.6 equals $S_1^{y_{\nu_0}} \geq k - 1 + l$. We have used the fact that $\#_{y_\nu}(\hat{M}_{q'}(t_1[u_2 \leftarrow p_2])) \geq 1$ for all ν and q', which follows from Lemma 3.11(1) because M is nondeleting (and hence so is $\hat{M}$).

The substitution $[\![\ldots]\!]$ is nondeleting, because $\hat{M}$ is nondeleting. Thus, since t has a subtree $\langle\!\langle r_1, p_1\rangle\!\rangle(\xi_1, \ldots, \xi_{m_1})$, it follows from Lemma 2.1 that $\hat{M}_{r_0}(t_0[u_1u_2 \leftarrow p_2]) = t[\![\ldots]\!]$ has a subtree $\langle\!\langle r_1, p_1\rangle\!\rangle(\xi_1, \ldots, \xi_{m_1})[\![\ldots]\!] = \hat{M}_{r_1}(t_1[u_2 \leftarrow p_2])[\ldots]$, where $[\ldots]$ denotes $[y_j \leftarrow \xi_j[\![\ldots]\!] \mid j \in [m_1]]$.

By the second assumption, $\hat{M}_{r_1}(t_1[u_2 \leftarrow p_2])$ has a subtree $\langle\!\langle r_2, p_2\rangle\!\rangle(\zeta_1, \ldots, \zeta_{m_2})$ with $\#_{y_{\nu_1}}(\zeta_{\nu_2}) \geq 1$. Thus we obtain a subtree $\langle\!\langle r_2, p_2\rangle\!\rangle(\zeta_1[\ldots], \ldots, \zeta_{m_2}[\ldots])$ and $\zeta_{\nu_2}[\ldots]$ has a subtree $\xi_{\nu_1}[\![\ldots]\!]$ which contains y_{ν_0} (because $\#_{y_{\nu_0}}(\xi_{\nu_1}) \geq 1$ and M is nondeleting). □

LEMMA 6.2. *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be a nondeleting MTT^R_{fnest} with the property: if $\langle\!\langle q, p\rangle\!\rangle \in \langle\!\langle Q, P\rangle\!\rangle^{(m)}$ is not reachable, then $\#_{y_j}(M_q(s)) \leq 1$ for all $j \in [m]$ and $s \in L_p$ (property* (ii) *of Definition* 5.6 *of p-properness).*

If M is not fcp, then there are $m \geq 1$, $q \in Q^{(m)}$, $j \in [m]$, $s \in T_\Sigma$, $u \in V(s)$, and $p \in P$ such that

(1) $\#_{y_j}(\hat{M}_q(s[u \leftarrow p])) \geq 2$,

(2) $\hat{M}_q(s[u \leftarrow p])$ *has a subtree* $\langle\!\langle q, p\rangle\!\rangle(\xi_1, \ldots, \xi_m)$ *with* $\#_{y_j}(\xi_j) \geq 1$, *and*

(3) $p = h(s) = h(s/u)$.

Proof. We first define an auxiliary notion. For $t \in T_\Sigma$, u an ancestor of $v \in V(t)$, $q \in Q^{(m)}$, $\mu \in [m]$, $q' \in Q^{(m')}$, $\mu' \in [m']$, define $(q, \mu) \to_{u,v} (q', \mu')$ if, for $\xi_{q,u,v} = \hat{M}_q(t/u[v' \leftarrow p_v])$ with $v = uv'$ and $p_v = h(t/v)$, $\#_{y_\mu}(\xi_{q,u,v}) \geq 2$ and $\xi_{q,u,v}$ has a subtree $\langle\!\langle q', p_v\rangle\!\rangle(\xi_1, \ldots, \xi_{m'})$ such that $\#_{y_\mu}(\xi_{\mu'}) \geq 1$. Note that $(q, \mu) \to_{u,v} (q', \mu')$ if and only if $\mathcal{P}(q, \mu, t/u, q', \mu', v', 2)$, where $\mathcal{P}$ is the statement of Lemma 6.1. The relation $\to$ is transitive; i.e., for a descendant w of v,

$$\text{if } (q, \mu) \to_{u,v} (q', \mu') \text{ and } (q', \mu') \to_{v,w} (q'', \mu''), \text{ then } (q, \mu) \to_{u,w} (q'', \mu'').$$

This follows from Lemma 6.1, because $(q, \mu) \to_{u,v} (q', \mu')$ and $(q', \mu') \to_{v,w} (q'', \mu'')$ imply that $\mathcal{P}(q, \mu, t/u, q'', \mu'', v'w', 3)$ with $w' \in \mathbb{N}^*$ such that $w = vw'$, and thus $(q, \mu) \to_{u,w} (q'', \mu'')$.

Assume that M is not fcp. Then, in terms of the $\to$ notation, the lemma says that there are $m \geq 1$, $q \in Q^{(m)}$, $j \in [m]$, $s \in T_\Sigma$, $u \in V(s)$, and $p \in P$ such that

- $(q, j) \to_{\varepsilon,u} (q, j)$ (points (1) and (2) of the lemma),
- $p = h(s) = h(s/u)$ (point (3) of the lemma).

Since M is not fcp, for every $n \in \mathbb{N}$, there are $q \in Q^{(m)}$, $j \in [m]$, and $t \in T_\Sigma$ such that $\#_{y_j}(M_q(t)) > n$. The following claim shows that if $\#_{y_j}(M_q(t/u))$ is "large" for a node u of t, then there must be a descendant v of u, a state r, and a parameter y_ν of r such that $(q, j) \to_{u,v} (r, \nu)$ and $\#_{y_\nu}(M_r(t/v))$ is still "large." The application of this claim can be iterated to show the existence of a sequence of descendants v and a sequence of steps $\to$, which will eventually lead to a repetition of a state-parameter pair that allows us to define s and u such that (1)–(3) hold.

Let B be a nesting bound for M. Let η be the maximal height of the right-hand side of a rule of M, i.e., $\eta = \max\{\text{height}(\text{rhs}(\rho)) \mid \rho \in R\}$, and let $\kappa \geq 1$ be an upper bound for the number of occurrences of one particular parameter in the right-hand side of a rule of M, i.e., $\#_y(\text{rhs}(\rho)) \leq \kappa$ for every $y \in Y$ and $\rho \in R$.

Claim. For every $c \geq 1$, $t \in T_\Sigma$, $u \in V(t)$, $q \in Q^{(m)}$, and $\mu \in [m]$, if $\#_{y_\mu}(M_q(t/u))$ is greater than $c^{B\eta} \cdot \kappa^B$, then there exist a descendant v of u, a state $r \in Q^{(m')}$, and a $\nu \in [m']$ such that $(q,\mu) \to_{u,v} (r,\nu)$ and $\#_{y_\nu}(M_r(t/v)) > c$.

Proof of the claim. Let w be a longest descendant of u such that $\#_{y_\mu}(\xi_{q,u,w}) = 1$. Clearly, such a w exists, because $\#_{y_\mu}(\xi_{q,u,u}) = 1$. Then there must be a child v of w that satisfies the requirements of the claim. Assume to the contrary, that if v is a child of w, then it does *not* satisfy the requirements, i.e., for every $r \in Q^{(m')}$ and $\nu \in [m']$ with $(q,\mu) \to_{u,v} (r,\nu)$, $\#_{y_\nu}(M_r(t/v)) \leq c$. This will lead to a contradiction.

By Lemmas 4.2 (applied to t/u and w) and 3.5,

$$M_q(t/u) = \xi_{q,u,w}[\![\text{rhs}]\!][\![\ldots]\!],$$

where $[\![\text{rhs}]\!] = [\![\langle\!\langle r, p_w\rangle\!\rangle \leftarrow \text{rhs}_M(r,\sigma,\langle p_1,\ldots,p_k\rangle) \mid r \in Q]\!]$ with $\sigma = t[w] \in \Sigma^{(k)}$, $k \geq 0$, $p_w = h(t/w)$, $p_i = h(t/wi)$ for $i \in [k]$, and $[\![\ldots]\!] = [\![\langle r, x_i\rangle \leftarrow M_r(t/wi) \mid \langle r, x_i\rangle \in \langle Q, X_k\rangle]\!]$. Now, $\#_{y_\mu}(\xi_{q,u,w}[\![\text{rhs}]\!]) \leq \kappa^B$. This is true because by Lemma 2.6, $\#_{y_\mu}(\xi_{q,u,w}[\![\text{rhs}]\!]) = S_1^{y_\mu} = \sum_{z \in V_{y_\mu}(\xi_{q,u,w})} \prod F^{[\![\text{rhs}]\!]}_{\xi_{q,u,w},z}$, which equals $\prod F^{[\![\text{rhs}]\!]}_{\xi_{q,u,w},z}$ for the unique z with $V_{y_\mu}(\xi_{q,u,w}) = \{z\}$. Since $\#_{y_\mu}(M_q(t/u)) > 1$, $\langle\!\langle q, h(t/u)\rangle\!\rangle$ is reachable by the assumption of the lemma. Thus, by Lemma 4.13, there are at most B occurrences of elements of $\langle\!\langle Q, \{p_w\}\rangle\!\rangle$ on the label path $\text{lpath}(\xi_{q,u,w}, z)$. Hence, $\prod F^{[\![\text{rhs}]\!]}_{\xi_{q,u,w},z}$ is the product of at most B numbers $\#_{y_\nu}(\text{rhs}_M(r,\sigma,\langle p_1,\ldots,p_k\rangle)) \leq \kappa$ for $r \in Q$ and $\nu \in [\text{rank}_Q(r)]$, and therefore $\prod F^{[\![\text{rhs}]\!]}_{\xi_{q,u,w},z} \leq \kappa^B$.

Since every label path of $\xi_{q,u,w}$ is of the form $w_0\langle\!\langle q_1, p_w\rangle\!\rangle w_1 \cdots \langle\!\langle q_l, p_w\rangle\!\rangle w_l$ with $l \leq B$, $q_1,\ldots,q_l \in Q$, and $w_0,\ldots,w_l \in \Delta^*$, it follows from Lemma 2.3(i) that every label path π in $\xi_{q,u,w}[\![\text{rhs}]\!]$ is of the form $w_0v_1w_1\cdots v_lw_l$, where each v_i is a label path in $\text{rhs}_M(q_i,\sigma,\langle p_1,\ldots,p_k\rangle)$. By the definition of η, the length of v_i is $\leq \eta$. Thus, $\#_{\langle Q,X_k\rangle}(\pi) = \sum_{i\in[l]} \#_{\langle Q,X_k\rangle}(v_i) \leq B\eta$.

Let $\zeta = \xi_{q,u,w}[\![\text{rhs}]\!]$. By Lemma 2.6, $\#_{y_\mu}(\zeta[\![\ldots]\!]) = \sum_{z\in V_{y_\mu}(\zeta)} \prod F^{[\![\ldots]\!]}_{\zeta,z}$. This is $\leq \kappa^B \cdot \prod F^{[\![\ldots]\!]}_{\zeta,z}$, where $z \in V_{y_\mu}(\zeta)$ such that $\prod F^{[\![\ldots]\!]}_{\zeta,z}$ is maximal, because $\#_{y_\mu}(\zeta) \leq \kappa^B$. Since $\#_{\langle Q,X_k\rangle}(\pi) \leq B\eta$ for $\pi = \text{lpath}(\zeta,z)$, $\prod F^{[\![\ldots]\!]}_{\zeta,z}$ is the product of at most $B\eta$ numbers $\#_{y_\nu}(M_r(t/wi))$. Let us now show that each such number is $\leq c$. We need to show that $(q,\mu) \to_{u,wi} (r,\nu)$. By the definition of w, $\#_{y_\mu}(\xi_{q,u,wi}) \neq 1$. Since M is nondeleting it follows from Lemma 3.11(1) that $\#_{y_\mu}(\xi_{q,u,wi}) \geq 1$, and thus $\#_{y_\mu}(\xi_{q,u,wi}) \geq 2$. Since $\langle r, x_i\rangle$ occurs in ζ at some node z' with $z = z'\nu z''$, ζ has a subtree $\langle r, x_i\rangle(\zeta_1,\ldots,\zeta_{m'})$ for some $\zeta_1,\ldots,\zeta_{m'} \in T_{\langle Q,X_k\rangle\cup\Delta}(Y_m)$, and y_μ occurs in ζ_ν. By Lemma 4.3, $\xi_{q,u,wi} = \zeta[\![..]\!][\![i]\!]$, with $[\![..]\!]$ and $[\![i]\!]$ as in that lemma. It follows from Lemma 3.11(1) that $[\![..]\!][\![i]\!]$ is nondeleting. Thus, by Lemma 2.1, $\xi_{q,u,wi}$ has a subtree $\langle\!\langle r, p_i\rangle\!\rangle(\zeta_1[\![..]\!][\![i]\!],\ldots,\zeta_{m'}[\![..]\!][\![i]\!])$ and y_μ occurs in $\zeta_\nu[\![..]\!][\![i]\!]$. This proves that $(q,\mu) \to_{u,wi} (r,\nu)$ and thus, by assumption, $\#_{y_\nu}(M_r(t/wi)) \leq c$. We get $\#_{y_\mu}(M_q(t/u)) \leq c^{B\eta} \cdot \kappa^B$, which is a contradiction and ends the proof of the claim.

Now, let $c_0 = 1$ and $c_i = c_{i-1}^{B\eta}\kappa^B$ for $i \geq 1$. Since M is not fcp, for every $n \geq 1$ there exist $r_0 \in Q^{(m_0)}$, $\nu_0 \in [m_0]$, and $t \in T_\Sigma$ such that $\#_{y_{\nu_0}}(M_{r_0}(t)) > c_n$. Let $v_0 = \varepsilon$. We apply the claim for $i = 0,1,\ldots,n-1$ to $u = v_i$, $q = r_i$, and $\mu = \nu_i$ to obtain that there exist a descendant v_{i+1} of v_i, a state $r_{i+1} \in Q^{(m_{i+1})}$, and $\nu_{i+1} \in [m_{i+1}]$ such that $(r_i,\nu_i) \to_{v_i,v_{i+1}} (r_{i+1},\nu_{i+1})$ and $\#_{y_{\nu_{i+1}}}(M_{r_{i+1}}(t/v_{i+1})) > c_{n-(i+1)}$.

Take $n = |Q| \cdot \overline{m} \cdot |P|$, where $\overline{m}$ is the maximal rank of a state of M. Then there are indices $0 \leq i < i' \leq n$ such that $q = r_i = r_{i'}$, $j = \nu_i = \nu_{i'}$, and $p = h(t/v_i) = h(t/v_{i'})$. Then $(q, j) \to_{v_i, v_{i'}} (q, j)$ by the transitivity of $\to$. Let $s = t/v_i$ and $v_i u = v_{i'}$. Clearly (3) holds. Moreover, in s, $(q, j) \to_{\varepsilon, u} (q, j)$, which means that (1) and (2) hold. ∎

We now prove that if a proper $\mathrm{MTT}^{\mathrm{R}}_{\mathrm{fnest}}$ M is lsi, then it is fcp; i.e., we prove step (II). The idea is to assume that M is not fcp, and then to "pump" the tree $s[u \leftarrow p]$ of Lemma 6.2 in order to show that this implies that M is not lsi. We use the following notation to pump a tree. For $s \in T_\Sigma$, $u \in V(s)$, $p \in P$, and $s' \in T_\Sigma(P)$, let $s[u \leftarrow p] \bullet s'$ denote $s[u \leftarrow s']$. Let $(s[u \leftarrow p])^0 = p$, and for $n \in \mathbb{N}$ let $(s[u \leftarrow p])^{n+1} = (s[u \leftarrow p]) \bullet (s[u \leftarrow p])^n$. Thus, e.g.,

$$\begin{aligned}
&(s[u \leftarrow p])^1 = s[u \leftarrow p] \bullet p = s[u \leftarrow p],\\
&(s[u \leftarrow p])^2 = (s[u \leftarrow p]) \bullet (s[u \leftarrow p]) = s[u \leftarrow s[u \leftarrow p]], \quad \text{and}\\
&(s[u \leftarrow p])^3 = (s[u \leftarrow p]) \bullet s[u \leftarrow s[u \leftarrow p]] = s[u \leftarrow s[u \leftarrow s[u \leftarrow p]]].
\end{aligned}$$

We will only pump the tree $s[u \leftarrow p]$, for a given $\mathrm{MTT}^{\mathrm{R}}$, if $\hat{h}(s[u \leftarrow p]) = p$. Note that this condition is satisfied in Lemma 6.2 by point (3).

THEOREM 6.3. *Let M be a proper MTT^R_{fnest}. If M is lsi, then it is fcp.*

Proof. Let $M = (Q, \Sigma, \Delta, q_0, R, P, h)$ be lsi; i.e., there is a $c \in \mathbb{N}$ such that for every input tree t,

$$(*) \qquad \mathrm{size}(\tau_M(t)) \leq c \cdot \mathrm{size}(t).$$

Assume now that M is not fcp. We will derive a contradiction by constructing an input tree t such that $\mathrm{size}(\tau_M(t)) > c \cdot \mathrm{size}(t)$. Let $q \in Q^{(m)}$, $m \geq 1$, $j \in [m]$, $s \in T_\Sigma$, $p = h(s)$, and $u \in V(s)$ be such that (1)–(3) of Lemma 6.2 hold. Note that since M is proper it satisfies the conditions of Lemma 6.2.

The idea of constructing a t such that $(*)$ does not hold is as follows: Let $s_0 \in T_\Sigma$ and $u_0 \in V(s_0)$ such that

$$(\dagger) \qquad \hat{M}_{q_0}(s_0[u_0 \leftarrow p]) \text{ has a subtree } \langle\!\langle q, p \rangle\!\rangle(\xi_1, \ldots, \xi_m)$$

for some trees $\xi_1, \ldots, \xi_m$. Consider input trees t_i obtained by i times pumping the tree $s[u \leftarrow p]$ in the tree $s_0[u_0 \leftarrow s]$. Then the size of the trees t_i grows at most linearly with constant $\mathrm{size}(s[u \leftarrow p])$. In the output tree $\tau_M(t_i)$ there are at least i occurrences of the subtree $\xi_j[\![\ldots]\!]$ for some second-order tree substitution $[\![\ldots]\!]$. Hence, the size of the trees $\tau_M(t_i)$ grows at least linearly with constant $\mathrm{size}(\xi_j)$. Thus, if we choose s_0 and u_0 in such a way that $\mathrm{size}(\xi_j)$ is larger than the product of c and $\mathrm{size}(s[u \leftarrow p])$, then $\mathrm{size}(\tau_M(t_i))$ grows faster than $c \cdot \mathrm{size}(t_i)$, which implies that we can find an i such that $(*)$ does not hold for $t = t_i$.

Recall Definition 5.6 of p-properness. In order to choose s_0 and u_0 appropriately we need the set $\mathrm{Arg}(q, j, p)$ to be infinite, i.e., to contain arbitrarily large trees. This is guaranteed by point (i) of Definition 5.6 if $\langle\!\langle q, p \rangle\!\rangle$ is reachable. The latter holds for the following reason: Since M is nondeleting, by Lemma 3.11(1), $\#_{y_\nu}(M_r(s/u)) \geq 1$ for every $r \in Q^{(m')}$ and $\nu \in [m']$. By Lemmas 4.2 and 2.6 and the fact that $\#_{y_j}(\hat{M}_q(s[u \leftarrow p])) \geq 2$ by (1), this implies that $\#_{y_j}(M_q(s)) \geq 2$. Thus, $\langle\!\langle q, p \rangle\!\rangle$ is reachable by point (ii) of Definition 5.6.

We now show the effect of pumping the tree $s[u \leftarrow p]$ in the input tree $s = s[u \leftarrow p] \bullet s/u$. For $i \geq 0$ let $t'_i = (s[u \leftarrow p])^i \bullet s/u$. Then $\#_{y_j}(M_q(t'_i)) > i$. Using the fact that M is nondeleting, this follows (as above, by Lemmas 4.2 and 2.6) from

$\#_{y_j}(\hat{M}_q(t'_i[u^i \leftarrow p])) > i$, which is a consequence of the next claim and the definition of $\mathcal{P}$ (cf. Lemma 6.1).

Claim. For $i \geq 0$, $\mathcal{P}(q, j, t'_i, q, j, u^i, i+1)$.

The proof of this claim is by induction on i. For $i = 0$, $\mathcal{P}(q, j, t'_i, q, j, u_1^i, i+1)$ because $\xi = \hat{M}_q(s/u[\varepsilon \leftarrow p]) = \langle\!\langle q, p\rangle\!\rangle(y_1, \ldots, y_m)$ and thus $\#_{y_j}(\xi) \geq 1$ and ξ has a subtree $\langle\!\langle q, p\rangle\!\rangle(\xi_1, \ldots, \xi_m)$ with $\#_{y_j}(\xi_j) = \#_{y_j}(y_j) = 1$. For $i + 1 > 0$, by induction, $\mathcal{P}(q, j, t'_i, q, j, u^i, i+1)$. Clearly, by (3), $h(t'_{i+1}/u^i) = h(s) = p = h(s/u) = h(t'_i/u^i)$, and $t'_{i+1}[u^i \leftarrow p] = t'_i[u^i \leftarrow p]$. Thus, $\mathcal{P}(q, j, t'_{i+1}, q, j, u^i, i+1)$. By (1) and (2), $\mathcal{P}(q, j, s, q, j, u, 2)$, which is equivalent to $\mathcal{P}(q, j, t'_{i+1}/u^i, q, j, u, 2)$ because $t'_{i+1}/u^i = s$. By Lemma 6.1 this means that $\mathcal{P}(q, j, t'_{i+1}, q, j, u^i u, i+2)$, which concludes the proof of the claim.

Now let $t_i = s_0[u_0 \leftarrow t'_i]$, where $s_0 \in T_\Sigma$ and $u_0 \in V(s_0)$ satisfy (†). Thus, t_i is the result of pumping the tree $s[u \leftarrow p]$ in the input tree $s_0[u_0 \leftarrow s]$. Since $\#_{y_j}(\hat{M}_q(t'_i) > i$, we obtain $\mathrm{size}(\tau_M(t_i)) > i \cdot \mathrm{size}(\xi_j)$ as follows: By Lemma 4.2, $\tau_M(t_i) = \hat{M}_{q_0}(s_0[u_0 \leftarrow p])[\![\ldots]\!]$, where $[\![\ldots]\!] = [\![\langle\!\langle r, p\rangle\!\rangle \leftarrow M_r(t'_i) \mid r \in Q]\!]$. By Lemma 2.1, $\hat{M}_{q_0}(s_0[u_0 \leftarrow p])[\![\ldots]\!]$ has a subtree $\xi = \langle\!\langle q, p\rangle\!\rangle(\xi_1, \ldots, \xi_m)[\![\ldots]\!] = M_q(t'_i)[y_\nu \leftarrow \xi_\nu[\![\ldots]\!] \mid \nu \in [m]]$. By Lemma 2.4 (summing for all $\delta \in \Delta$), $\mathrm{size}(\xi) = \#_\Delta(\xi) = \#_\Delta(M_q(t'_i)) + \sum_{\nu \in [m]} \#_{y_\nu}(M_q(t'_i)) \cdot \#_\Delta(\xi_\nu[\![\ldots]\!]) \geq \sum_{\nu=j} \#_{y_\nu}(M_q(t'_i)) \cdot \#_\Delta(\xi_\nu[\![\ldots]\!]) = \#_{y_j}(M_q(t'_i)) \cdot \mathrm{size}(\xi_j[\![\ldots]\!])$. Since M is productive, Lemmas 2.7 and 3.11 imply that $\mathrm{size}(\xi_j[\![\ldots]\!]) \geq \mathrm{size}(\xi_j)$. Since $\#_{y_j}(M_q(t'_i)) > i$, this implies that $\mathrm{size}(\tau_M(t_i)) > i \cdot \mathrm{size}(\xi_j)$.

Since $\mathrm{Arg}(q, j, p)$ is infinite, we can choose s_0 and u_0 such that (†) and

$$\mathrm{size}(\xi_j) > c \cdot c_1,$$

where $c_1 = \mathrm{size}(s[u \leftarrow p]) - 1$. Let $i = c(c_0 + c_2)$ for $c_0 = \mathrm{size}(s_0[u_0 \leftarrow p]) - 1$ and $c_2 = \mathrm{size}(s/u)$. Since $\mathrm{size}(t_i) = c_0 + ic_1 + c_2$ this means that $\mathrm{size}(\tau_M(t_i)) > c \cdot \mathrm{size}(t_i)$ because $\mathrm{size}(\tau_M(t_i)) > i \cdot \mathrm{size}(\xi_j) \geq i \cdot (c \cdot c_1 + 1) = icc_1 + c(c_0 + c_2) = c(c_0 + ic_1 + c_2) = c \cdot \mathrm{size}(t_i)$. This contradicts $(*)$ and concludes the proof. □

6.2. From lsi, fnest, and fcp to fci (III). Here we present a pumping lemma for $\mathrm{MTT}^{\mathrm{R}}_{\mathrm{fnest,fcp}}$s that are not fci (Lemma 6.5) and apply it in Lemma 6.6 to show that if an $\mathrm{MTT}^{\mathrm{R}}_{\mathrm{fnest,fcp}}$ is lsi, then it is fci. We first define, in general, what is required of an $\mathrm{MTT}^{\mathrm{R}}$ in order to get a repetition of states by pumping a part of an input tree; this is called *input pumpable*. It means that there is a state q_1 that is reachable, i.e., appears in $\hat{M}_{q_0}(s_0[u_0 \leftarrow p])$ for some input tree s_0 and node u_0 of s_0 (with $p = h(s_0/u_0)$), and going from node u_0 to node u_0u_1 in s_0, q_1 will generate a copy of itself and of a state q_2; furthermore, the state q_2 generates a copy of itself when going from u_0 to u_0u_1.

Definition 6.4 (input pumpable). *An MTT^R $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ is* input pumpable *if there are $q_1, q_2 \in Q$, $s_0 \in T_\Sigma$, $u_0 \in V(s_0)$, $u_1 \in V(s_0/u_0)$, and $p \in P$ such that the following four conditions hold:*

(1) *$\langle\!\langle q_1, p\rangle\!\rangle$ occurs in $\hat{M}_{q_0}(s_0[u_0 \leftarrow p])$,*

(2) *$\langle\!\langle q_1, p\rangle\!\rangle$ and $\langle\!\langle q_2, p\rangle\!\rangle$ occur at distinct nodes of $\hat{M}_{q_1}(s_0/u_0[u_1 \leftarrow p])$,*

(3) *$\langle\!\langle q_2, p\rangle\!\rangle$ occurs in $\hat{M}_{q_2}(s_0/u_0[u_1 \leftarrow p])$, and*

(4) *$p = h(s_0/u_0) = h(s_0/u_0u_1)$.*

The following pumping lemma can be viewed as a generalization of Lemma 4.2 of [1] from top-down tree transducers to MTTs.

Lemma 6.5. *Let M be a nondeleting $MTT^R_{fnest,fcp}$. If M is not fci, then it is input pumpable.*

Proof. Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$. We first define some auxiliary notions. Let $t \in T_\Sigma$ and $u, v \in V(t)$ such that u is an ancestor of v, i.e., $v = uv'$ for some $v' \in \mathbb{N}^*$,

and let $p_v = h(t/v)$. For $q \in Q$, if $n = \#_{\langle\langle Q,\{p_v\}\rangle\rangle}(\hat{M}_q(t/u[v' \leftarrow p_v]))$, then we say that q *contributes* n *states at* u *to* v. If $n \geq 1$, then we say that q contributes at u to v. For $q, q' \in Q$ we write $q \to_{u,v} q'$ if $\langle\langle q', p_v\rangle\rangle$ occurs in $\hat{M}_q(t/u[v' \leftarrow p_v])$. For $r_1, r_2 \in Q$ we write $q \to_{u,v} r_1, r_2$ if $\langle\langle r_1, p_v\rangle\rangle$ and $\langle\langle r_2, p_v\rangle\rangle$ occur at distinct nodes of $\hat{M}_q(t/u[v' \leftarrow p_v])$. Observe the following easy properties:

(P0) $q \to_{v,v} q'$ if and only if $q = q'$; q contributes one state at v to v.

(P1) $q_0 \to_{\varepsilon,v} q$ if and only if q occurs in $\mathrm{sts}_M(t, v)$; q_0 contributes $|\mathrm{sts}_M(t, v)|$ states at ε to v.

(P2) q contributes at u to v if and only if there is a $q' \in Q$ such that $q \to_{u,v} q'$.

Let w be a node of t that is a descendant of u and an ancestor of v.

(P3) If $q \to_{u,w} q''$ and $q'' \to_{w,v} q'$, then $q \to_{u,v} q'$.

(P4) If $q \to_{u,v} q'$, then there is a $q'' \in Q$ such that $q \to_{u,w} q''$ and $q'' \to_{w,v} q'$.

Note that (P3) and (P4) can be proved using Lemma 4.14: Let $w', v'' \in \mathbb{N}^*$ such that $w = uw'$ and $v = wv''$ (and so v' above equals $w'v''$), and let $p_w = h(t/w)$. For (P3), the number $\#_{\langle\langle q'', p_w\rangle\rangle}(\hat{M}_q(t/u[w' \leftarrow p_w]))$ is ≥ 1 because $q \to_{u,w} q''$, and $\#_{\langle\langle q', p_v\rangle\rangle}(\hat{M}_{q''}(t/w[v'' \leftarrow p_v]))$ is ≥ 1 because $q'' \to_{w,v} q'$; hence the product of these two numbers is ≥ 1 and so the sum S of Lemma 4.14 is ≥ 1. Thus, by part (i) of that lemma, $\#_{\langle\langle q', p_v\rangle\rangle}(\hat{M}_q(t/u[v' \leftarrow p_v])) \geq 1$, i.e., $q \to_{u,v} q'$. For (P4), $q \to_{u,v} q'$ implies that the sum in $(\times)$ of the proof of Lemma 4.14 is ≥ 1, and thus there is an occurrence of some $\langle\langle q'', p_w\rangle\rangle \in \langle\langle Q, \{p_w\}\rangle\rangle$ in $\hat{M}_q(s/u[w' \leftarrow p_w])$ with $\#_{\langle\langle q', p_v\rangle\rangle}(\hat{M}_{q''}(t/w[v'' \leftarrow p_v])) \geq 1$; i.e., there is a $q'' \in Q$ such that $q \to_{u,w} q''$ and $q'' \to_{w,v} q'$.

(P5) q contributes ≥ 2 states at u to v if and only if there are $r_1, r_2 \in Q$ such that $q \to_{u,v} r_1, r_2$.

(P6) Let $r_1', r_2' \in Q$ and w as above. If $q \to_{u,w} r_1, r_2$ and $r_i \to_{w,v} r_i'$ for $i \in [2]$, then $q \to_{u,v} r_1', r_2'$.

Let us prove property (P6). If $r_1' \neq r_2'$, then by (P3), $q \to_{u,v} r_1'$ and $q \to_{u,v} r_2'$, which means that $q \to_{u,v} r_1', r_2'$. Now assume that $r_1' = r_2'$. By Lemma 4.14(i), $\#_{\langle\langle r_1', p_v\rangle\rangle}(\hat{M}_q(t/u[v' \leftarrow p_v]))$ is greater than or equal to

$$(*) \qquad \sum_{r \in Q} \#_{\langle\langle r_1', p_v\rangle\rangle}(\hat{M}_r(t/w[v'' \leftarrow p_v])) \cdot \#_{\langle\langle r, p_w\rangle\rangle}(\hat{M}_q(t/u[w' \leftarrow p_w])),$$

where p_w, w', and v'' are as in the proof of (P3). We distinguish the following two cases:

(i) $r_1 \neq r_2$: For $r = r_1$ and $r = r_2$, $\#_{\langle\langle r, p_w\rangle\rangle}(\hat{M}_q(t/u[w' \leftarrow p_w])) \geq 1$, because $q \to_{u,w} r_1, r_2$. Thus, the sum in $(*)$ is $\geq \#_{\langle\langle r_1', p_v\rangle\rangle}(\hat{M}_{r_1}(t/w[v'' \leftarrow p_v])) + \#_{\langle\langle r_1', p_v\rangle\rangle}(\hat{M}_{r_2}(t/w[v'' \leftarrow p_v]))$, which is ≥ 2, because $r_i \to_{w,v} r_i'$ for $i \in [2]$.

(ii) $r_1 = r_2$: For $r = r_1$, $\#_{\langle\langle r, p_w\rangle\rangle}(\hat{M}_q(t/u[w' \leftarrow p_w])) \geq 2$, because $q \to_{u,w} r_1, r_1$. Thus, the sum in $(*)$ is $\geq \#_{\langle\langle r_1', p_v\rangle\rangle}(\hat{M}_{r_1}(t/w[v'' \leftarrow p_v])) \cdot 2$, which is ≥ 2, because $r_1 \to_{w,v} r_1'$.

In terms of the $\to$ notation the four conditions of input pumpability (cf. Definition 6.4) say that there are states q_1 and q_2, a tree $s_0 \in T_\Sigma$, and nodes u_0 and u_0u_1 of s_0 such that

(1) $q_0 \to_{\varepsilon,u_0} q_1$,

(2) $q_1 \to_{u_0,u_0u_1} q_1, q_2$,

(3) $q_2 \to_{u_0,u_0u_1} q_2$, and

(4) $h(s_0/u_0) = h(s_0/u_0u_1)$.

Since M is not fci, arbitrary long state sequences can be generated. Thus, for every $m \geq 1$ there are $t \in T_\Sigma$ and $v \in V(t)$ such that $|\mathrm{sts}_M(t, v)| > m$, which, by (P1),

means that q_0 contributes more than m states at ε to v. In Claim 1 below we will show that if a state q contributes "many" states at u to v, then there must be an intermediate node w (a descendant of u and ancestor of v) such that q contributes at least two states at u to w that contribute at w to v, and at least one of these states still contributes "many" states at w to v. The application of this claim can be iterated to show the existence of a sequence of intermediate nodes w, which will eventually lead to an appropriate repetition of states (and look-ahead states) that allows us to define s_0 and nodes u_0, u_0u_1 for which (1)–(4) hold.

Let $\kappa \geq 1$ be an upper bound for the number of occurrences of elements of $\langle Q, \{x_i\}\rangle$ for an $i \geq 1$ in the right-hand side of any rule of R, i.e., $\kappa \geq \#_{\langle Q,\{x_i\}\rangle}(\mathrm{rhs}(\rho))$ for every $\rho \in R$ and $i \geq 1$. Let η be the maximal height of the right-hand side of any rule in R, i.e., $\eta = \max\{\mathrm{height}(\mathrm{rhs}(\rho)) \mid \rho \in R\}$. Let $N \geq 1$ be a parameter copying bound for M and let $B \geq 1$ be a nesting bound for M.

Claim 1. Let $\langle\!\langle q, p\rangle\!\rangle \in \langle\!\langle Q, P\rangle\!\rangle$ be reachable, $t \in T_\Sigma$, and $u, v \in V(t)$ such that $t/u \in L_p$ and u is an ancestor of v. Let $c \geq 1$. If q contributes more than $(\kappa N^{2B+\eta}) \cdot c$ states at u to v, then there is a proper descendant w of u which is an ancestor of v and there are states $r, r' \in Q$ such that

(a) $q \to_{u,w} r, r'$,

(b) r contributes more than c states at w to v, and

(c) r' contributes at w to v.

Proof of Claim 1. Let w be the first (shortest) descendant of u and ancestor of v such that there are $r_1, r_2 \in Q$ with $q \to_{u,w} r_1, r_2$ and r_1,r_2 contribute at w to v. Clearly such a w exists, because q contributes ≥ 2 states at u to v, and thus, by (P5), there are $r_1, r_2 \in Q$ such that $q \to_{u,v} r_1, r_2$, and, by (P0), r_1, r_2 contribute at v to v. By (P0), q contributes exactly one state at u to u and therefore $w \neq u$. It remains to show that there is an $r \in Q$ such that $q \to_{u,w} r$ and r contributes more than c states at w to v; then r' is chosen to be one of the r_1, r_2 such that (a) holds.

In (sub)Claim 2 below we will show that q contributes at most $\kappa \cdot N^{B+\eta}$ states r at u to w that contribute at w to v. We now show that the number of states that q contributes at u to v is at most N^B times the sum of the contributions of the states r at w to v, and hence that at least one of these r must contribute $> c$ states.

Let $w', v', v'' \in \mathbb{N}^*$ such that $w = uw'$ and $v = uv' = wv''$. Let $p_w = h(t/w)$ and $p_v = h(t/v)$. By assumption, q contributes $> (\kappa N^{2B+\eta}) \cdot c$ states at u to v, i.e., $(\kappa N^{2B+\eta}) \cdot c$ is smaller than $\#_{\langle\!\langle Q,\{p_v\}\rangle\!\rangle}(\hat{M}_q(t/u[v' \leftarrow p_v]))$, which, by Lemma 4.14(ii) (using the fact that $\langle\!\langle q, h(t/u)\rangle\!\rangle$ is reachable, and summing over all elements of $\langle\!\langle Q, \{p_v\}\rangle\!\rangle$), is

$$\leq N^B \cdot \sum_{r \in Q} \#_{\langle\!\langle Q,\{p_v\}\rangle\!\rangle}(\hat{M}_r(t/w[v'' \leftarrow p_v])) \cdot \#_{\langle\!\langle r,p_w\rangle\!\rangle}(\hat{M}_q(t/u[w' \leftarrow p_w])).$$

If $\#_{\langle\!\langle Q,\{p_v\}\rangle\!\rangle}(\hat{M}_r(t/w[v'' \leftarrow p_v])) \neq 0$, then r contributes at w to v. Thus, we can restrict the above sum to states in $Q_{w,v} = \{r \in Q \mid r \text{ contributes at } w \text{ to } v\}$. Now let $r \in Q_{w,v}$ be such that $q \to_{u,w} r$ (i.e., $\#_{\langle\!\langle r,p_w\rangle\!\rangle}(\hat{M}_q(t/u[w' \leftarrow p_w])) \geq 1$) and the number of states it contributes at w to v is maximal; i.e., for all $r' \neq r$ with $q \to_{u,w} r'$, $\#_{\langle\!\langle Q,\{p_v\}\rangle\!\rangle}(\hat{M}_{r'}(t/w[v'' \leftarrow p_v])) \leq \#_{\langle\!\langle Q,\{p_v\}\rangle\!\rangle}(\hat{M}_r(t/w[v'' \leftarrow p_v]))$. Then the above number is

$$\leq N^B \cdot \#_{\langle\!\langle Q,\{p_v\}\rangle\!\rangle}(\hat{M}_r(t/w[v'' \leftarrow p_v])) \cdot \#_{\langle\!\langle Q_{w,v},\{p_w\}\rangle\!\rangle}(\hat{M}_q(t/u[w' \leftarrow p_w])),$$

which, by Claim 2, is $\leq N^B \cdot \#_{\langle\!\langle Q,\{p_v\}\rangle\!\rangle}(\hat{M}_r(t/w[v'' \leftarrow p_v])) \cdot (\kappa N^{B+\eta})$. Thus we get

$c < \#_{\langle\langle Q,\{p_v\}\rangle\rangle}(\hat{M}_r(t/w[v'' \leftarrow p_v]))$, i.e., r contributes more than c states at w to v, which concludes the proof of Claim 1.

Claim 2. $\#_{\langle\langle Q_{w,v},\{p_w\}\rangle\rangle}(\hat{M}_q(t/u[w' \leftarrow p_w])) \leq \kappa \cdot N^{B+\eta}$.

Proof of Claim 2. Since $w \neq u$ it follows that $w' \neq \varepsilon$; i.e., there are $i \geq 1$ and $\omega' \in \mathbb{N}^*$ such that $w' = \omega' i$. Let $\omega = u\omega'$; i.e., w is the ith child of ω. In the remainder of this proof we will always write ωi in place of w and $\omega' i$ in place of w', in particular, $p_{\omega i} = p_w$ and $Q_{\omega i,v} = Q_{w,v}$. Let $p_\omega = h(t/\omega)$. Using the fact that $\langle\langle q, h(t/u)\rangle\rangle$ is reachable, we can apply Lemma 4.14(ii) to t and $u, \omega, \omega i \in V(t)$, summing over all $\langle\langle q', p_{\omega i}\rangle\rangle$ in $\langle\langle Q_{\omega i,v}, \{p_{\omega i}\}\rangle\rangle$, to get that $\#_{\langle\langle Q_{\omega i,v},\{p_{\omega i}\}\rangle\rangle}(\hat{M}_q(t/u[\omega' i \leftarrow p_{\omega i}]))$ is

$$\leq N^B \cdot \sum_{r \in Q} \#_{\langle\langle Q_{\omega i,v},\{p_{\omega i}\}\rangle\rangle}(\hat{M}_r(t/\omega[i \leftarrow p_{\omega i}])) \cdot \#_{\langle\langle r,p_\omega\rangle\rangle}(\hat{M}_q(t/u[\omega' \leftarrow p_\omega])).$$

If $\#_{\langle\langle Q_{\omega i,v},\{p_{\omega i}\}\rangle\rangle}(\hat{M}_r(t/\omega[i \leftarrow p_{\omega i}])) \neq 0$, then there is an occurrence of some $\langle\langle r', p_{\omega i}\rangle\rangle$ in $\hat{M}_r(t/\omega[i \leftarrow p_{\omega i}])$, i.e., $r \rightarrow_{\omega,\omega i} r'$, and r' contributes at ωi to v, i.e., $r' \rightarrow_{\omega i,v} r''$ for some $r'' \in Q$. Thus, by (P3), $r \rightarrow_{\omega,v} r''$, which means by (P2) that r contributes at ω to v. By the definition of the node ωi there is at most one occurrence of a $\langle\langle q', p_\omega\rangle\rangle \in \langle\langle Q, \{p_\omega\}\rangle\rangle$ in $\hat{M}_q(t/u[\omega' \leftarrow p_\omega])$ such that q' contributes at ω to v, and since q contributes at u to v, by (P4) there is at least one such occurrence. Hence, in the above sum there is only one nonzero product, namely, for $r = q'$, and $\#_{\langle\langle q',\{p_\omega\}\rangle\rangle}(\hat{M}_q(t/u[\omega' \leftarrow p_\omega])) = 1$. We get

$$N^B \cdot \#_{\langle\langle Q_{\omega i,v},\{p_{\omega i}\}\rangle\rangle}(\hat{M}_{q'}(t/\omega[i \leftarrow p_{\omega i}])) \leq N^B \cdot \#_{\langle\langle Q,\{p_{\omega i}\}\rangle\rangle}(\hat{M}_{q'}(t/\omega[i \leftarrow p_{\omega i}])).$$

By Lemma 4.3 with $s = t/\omega$ and $u = \varepsilon$, and since $\hat{M}_{q'}(t/\omega[\varepsilon \leftarrow p_\omega]) = \langle\langle q', p_\omega\rangle\rangle$, the tree $\hat{M}_{q'}(t/\omega[i \leftarrow p_{\omega i}])$ equals $\mathrm{rhs}_M(q', \sigma, \langle p_1, \ldots, p_k\rangle)[\![..]\!][\![i]\!]$, where $[\![..]\!] = [\![\langle r', x_j\rangle \leftarrow M_{r'}(t/\omega j) \mid r' \in Q, j \in [k] - \{i\}]\!]$ and $[\![i]\!] = [\![\langle r', x_i\rangle \leftarrow \langle\langle r', p_{\omega i}\rangle\rangle \mid r' \in Q]\!]$ with $t[\omega] = \sigma \in \Sigma^{(k)}$, $k \geq 1$, and $p_j = h(t/\omega j)$ for each $j \in [k]$. Thus, $N^B \cdot \#_{\langle\langle Q,\{p_{\omega i}\}\rangle\rangle}(\hat{M}_{q'}(t/\omega[i \leftarrow p_{\omega i}]))$ equals $N^B \cdot \#_{\langle\langle Q,\{p_{\omega i}\}\rangle\rangle}(\mathrm{rhs}_M(q', \sigma, \langle p_1, \ldots, p_k\rangle)[\![..]\!][\![i]\!])$, which, avoiding the relabeling $[\![i]\!]$, can be written as

$$N^B \cdot \#_{\langle Q,\{x_i\}\rangle}(\mathrm{rhs}_M(q', \sigma, \langle p_1, \ldots, p_k\rangle)[\![\langle r', x_j\rangle \leftarrow M_{r'}(t/\omega j) \mid r' \in Q, j \neq i]\!]).$$

The application of Lemma 2.6 and the fact that the trees $M_{r'}(t/\omega j)$ do not contain elements of $\langle Q, \{x_i\}\rangle$ gives the number $N^B \cdot \sum_{\tilde{u} \in V_{\langle Q,\{x_i\}\rangle}(\zeta)} \prod F_{\zeta,\tilde{u}}^{[\![..]\!]}$, where $\zeta = \mathrm{rhs}_M(q', \sigma, \langle p_1, \ldots, p_k\rangle)$. Since the height of ζ is at most η, $\prod F_{\zeta,\tilde{u}}^{[\![..]\!]} \leq N^\eta$, and thus the above number is $\leq N^{B+\eta} \cdot |V_{\langle Q,\{x_i\}\rangle}(\zeta)|$, which is $\leq \kappa \cdot N^{B+\eta}$ by the definition of κ. This ends the proof of Claim 2.

Let $\gamma = \kappa N^{2B+\eta}$. Since M is not fci, for every $n \geq 1$ there are $t_n \in T_\Sigma$ and $v_n \in V(t_n)$ such that $|\mathrm{sts}_M(t_n, v_n)| > \gamma^n$. Let $r_0 = q_0$ and $w_0 = \varepsilon$. We now apply Claim 1 for $i = 0, \ldots, n-1$ to $q = r_i$, $p = h(t_n/w_i)$, $t = t_n$, $u = w_i$, $v = v_n$, and $c = \gamma^{n-i-1}$. For $i = 0$ this is possible because $\langle\langle q_0, h(t_n)\rangle\rangle$ is reachable, and by (P1), q_0 contributes more than γ^n states at ε to v_n. We obtain that there exists a proper descendant w_{i+1} of w_i and states r_{i+1}, r'_{i+1} such that $r_i \rightarrow_{w_i,w_{i+1}} r_{i+1}, r'_{i+1}$, the state r_{i+1} contributes more than γ^{n-i-2} states at w_{i+1} to v_n, and r'_{i+1} contributes at w_{i+1} to v_n. Note that since $q_0 \rightarrow_{\varepsilon,w_{i+1}} r_{i+1}$ and $q_0 \rightarrow_{\varepsilon,w_{i+1}} r'_{i+1}$ by (P3), both r_{i+1} and r'_{i+1} occur in $\mathrm{sts}_M(t_n, w_{i+1})$ by (P1) (and thus $\langle\langle r_{i+1}, h(t_n/w_{i+1})\rangle\rangle$ is reachable). For an ancestor w of v_n let $\mathrm{csts}(w)$ denote $\mathrm{sts}_M(t_n, w)$ restricted to the states q which contribute at w to v_n (i.e., all states that do not contribute to v_n are erased from

$\mathrm{sts}_M(t_n, w)$). Hence, r occurs in $\mathrm{csts}(w)$ if and only if $q_0 \to_{\varepsilon,w} r \to_{w,v} q$ for some state q. In particular, r_{i+1} and r'_{i+1} occur in $\mathrm{csts}(w_{i+1})$. Figure 6.1 shows the nodes w_i and the corresponding sequences $\mathrm{csts}(w_i)$ with the states r_i, r'_i; the arrows mean $\to_{w_i,w_{i+1}}$.

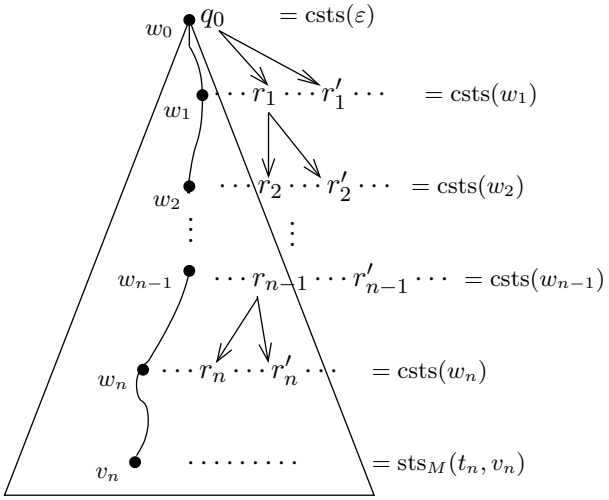


FIG. 6.1. *The tree t_n with contributing states.*

Now take $n = |Q| \cdot |P| \cdot 2^{|Q|}$ and let t_n, v_n, w_i, r_i, and r'_i be as above for $0 \le i \le n$. Clearly this means that there are indices $0 \le i < j \le n$ such that

- $r_i = r_j$,
- $p = h(t_n/w_i) = h(t_n/w_j)$, and
- $\{r \in Q \mid r \text{ occurs in } \mathrm{csts}(w_i)\} = \{r \in Q \mid r \text{ occurs in } \mathrm{csts}(w_j)\}$,

because there are exactly $|Q| \cdot |P| \cdot 2^{|Q|}$ different possibilities (r_i, p, S) for $r_i \in Q$, $p \in P$, and $S \subseteq Q$. Let $q'_1 = r_i$ and let $q'_2 \in Q$ such that $r'_{i+1} \to_{w_{i+1},w_j} q'_2$ and q'_2 occurs in $\mathrm{csts}(w_j)$. Such a q'_2 exists by the fact that r'_{i+1} contributes at w_{i+1} to v_n, using property (P4) (and also (P2) and (P3)). Since $r_{i+1} \to_{w_{i+1},w_j} r_i$, we can apply (P6) to get $q'_1 \to_{w_i,w_j} q'_1, q'_2$. Thus, conditions (1), (2), and (4) of input pumpability hold for $q_1 = q'_1$, $q_2 = q'_2$, $s_0 = t_n$, $u_0 = w_i$, and $u_0u_1 = w_j$. Clearly, if $q'_1 = q'_2$, then also (3) holds, which proves the lemma for that case. Thus, from now on we assume that $q'_1 \ne q'_2$. To realize (3), we will pump the tree $t_n/w_i[w'_j \leftarrow p]$ in t_n, where $w_j = w_i w'_j$.

For every $r \in Q$ that occurs in $\mathrm{csts}(w_i)$, there is an $r' \in Q$ with $r \to_{w_i,w_j} r'$, and r' occurs in $\mathrm{csts}(w_j)$ by (P4). Since the same states appear in $\mathrm{csts}(w_i)$ and $\mathrm{csts}(w_j)$, this means that r' also occurs in $\mathrm{csts}(w_i)$. Thus, there is a sequence

$$q'_1 \to_{w_i,w_j} q'_2 \to_{w_i,w_j} q'_3 \to_{w_i,w_j} \cdots \to_{w_i,w_j} q'_m \to_{w_i,w_j} q'_{m-\nu},$$

where $2 \le m \le |Q|$, $0 \le \nu < m$, and $q'_1, \ldots, q'_m$ are pairwise different states that occur in $\mathrm{csts}(w_i)$. Hence, after $m - \nu - 1$ steps of $\to_{w_i,w_j}$, starting at q'_1, states will repeat with period $\nu + 1$. Let d be a multiple of $\nu + 1$ with $d \ge m - \nu - 1$. Then there is a $\mu \in \{m - \nu, \ldots, m\}$ such that after d steps of $\to_{w_i,w_j}$, q'_1 reaches q'_μ and q'_μ reaches q'_μ.

Let $q_1 = q'_1$, $q_2 = q'_\mu$,

$$s_0 = (t_n[w_i \leftarrow p]) \bullet (t_n/w_i[w'_j \leftarrow p])^d \bullet (t_n/w_j),$$

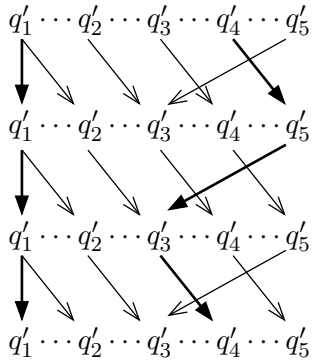


FIG. 6.2. *Conditions* (2) *and* (3) *of input pumpability for* $q_1 = q'_1$ *and* $q_2 = q'_4$.

$u_0 = w_i$, and $u_1 = (w'_j)^d$. Then $h(s_0/w_i(w'_j)^\gamma) = p$ for all $0 \le \gamma \le d$, which easily follows by induction, using the fact that $\hat{h}(t_n/w_i[w'_j \leftarrow p]) = \hat{h}(t_n/w_i[w'_j \leftarrow h(t_n/w_j)]) = h(t_n/w_i) = p$. In particular $h(s_0/u_0) = h(s_0/u_0u_1) = p$; i.e., condition (4) of input pumpability holds. Clearly, for $0 \le \gamma < d$, $q \to_{w_i,w_j} q'$ in the tree t_n if and only if $q \to_{w_i(w'_j)^\gamma, w_i(w'_j)^{\gamma+1}} q'$ in the tree s_0, and similarly $q \to_{w_i,w_j} q', q''$ in the tree t_n if and only if $q \to_{w_i(w'_j)^\gamma, w_i(w'_j)^{\gamma+1}} q', q''$ in the tree s_0; this is true because $s_0/w_i(w'_j)^\gamma[w'_j \leftarrow p] = t_n/w_i[w'_j \leftarrow p]$. Thus, in s_0, $q_2 \to_{u_0,u_0u_1} q_2$ by the definition of q'_μ (using (P3)), which proves condition (3) of the input pumpable property. To show condition (2) we use (P6): Since $q'_1 \to_{w_i,w_iw'_j} q'_1, q'_2$, also $q'_1 \to_{w_i,w_iw'_j} q'_1$ and thus, by the above and by (P3), $q'_1 \to_{w_iw'_j, w_i(w'_j)^d} q'_1$ holds in s_0. By the definition of q'_μ, $q'_2 \to_{w_iw'_j, w_i(w'_j)^d} q'_\mu$. Therefore, by (P6), $q_1 \to_{u_0,u_0u_1} q_1, q_2$. Clearly, (1) of input pumpability holds because $q_0 \to_{\varepsilon,w_i} r_i$ in t_n by the definition of r_i, $s_0[u_0 \leftarrow p] = t_n[w_i \leftarrow p]$, and consequently $q_0 \to_{\varepsilon,u_0} r_i = q_1$ holds in s_0. Figure 6.2 outlines the choice of q_2 for $m = 5$ and $\nu = 2$ (thus $d = 3$ and $\mu = 4$). □

LEMMA 6.6. *Let* M *be a proper* MTT^R. *If* M *is input pumpable, then it is not lsi.*

Proof. Let $M = (Q, \Sigma, \Delta, q_0, R, P, h)$ be input pumpable; i.e., there are $q_1, q_2 \in Q$, $s_0 \in T_\Sigma$, $u_0 \in V(s_0)$, $u_1 \in V(s_0/u_0)$, and $p \in P$ such that (1)–(4) of Definition 6.4 hold. Assume now that M is lsi; i.e., there is a $c \in \mathbb{N}$ such that for every input tree $t \in T_\Sigma$,

$$(*) \qquad \text{size}(\tau_M(t)) \le c \cdot \text{size}(t).$$

In what follows we will derive a contradiction by constructing an input tree t such that $\text{size}(\tau_M(t)) > c \cdot \text{size}(t)$. Note first that if we replace in s_0 the subtree at u_0u_1 by any tree s in L_p, then (1)–(4) still hold. Similar to the proof of Theorem 6.3, the idea of constructing t is as follows: Consider input trees t_i obtained by i times pumping the tree $s_0/u_0[u_1 \leftarrow p]$ in the tree $s_0[u_0u_1 \leftarrow s]$. Then the trees t_i grow at most linearly with constant $\text{size}(s_0/u_0[u_1 \leftarrow p])$. In the output tree $\tau_M(t_i)$ there are at least i occurrences of the tree $M_{q_2}(s)$. Hence, the trees $\tau_M(t_i)$ grow at least linearly with constant $\text{size}(M_{q_2}(s))$. Thus, if we choose s in such a way that $\text{size}(M_{q_2}(s))$ is larger than the product of c and the size of $s_0/u_0[u_1 \leftarrow p]$, then $\text{size}(\tau_M(t_i))$ grows faster than $c \cdot \text{size}(t_i)$; i.e., we can find an i such that $(*)$ does not hold for $t = t_i$.

In order to choose the tree s appropriately, we need the set $\text{Out}(q_2, p) = \{M_{q_2}(s) \mid s \in L_p\}$ to be infinite, i.e., to contain trees with arbitrarily many output symbols.

This is guaranteed by i-properness (cf. point (i) of Definition 5.1) if (a) $\langle\langle q_2, p\rangle\rangle$ is reachable and (b) $q_2 \neq q_0$.

(a) Clearly, $\langle\langle q_2, p\rangle\rangle$ is reachable because it occurs in $\hat{M}_{q_0}(s_0[u_0u_1 \leftarrow p])$; this follows from (1) and (2) using Lemma 4.14(i) (analogous to the proof of (P3) in the proof of Lemma 6.5; in fact, using the $\rightarrow$ notation of the proof of that lemma, it follows from (1) and (2) by (P3) that $q_0 \rightarrow_{\varepsilon,u_0u_1} q_2$, which means that $\langle\langle q_2, p\rangle\rangle$ occurs in $\hat{M}_{q_0}(s_0[u_0u_1 \leftarrow p])$).

(b) By (2), $\hat{M}_{q_1}(s_0/u_0[u_1 \leftarrow p]) \neq \langle\langle q_1, p\rangle\rangle = \hat{M}_{q_1}(s_0/u_0[\varepsilon \leftarrow p])$, and thus $u_1 \neq \varepsilon$; i.e., $u_1 = u_1'i$ for some $u_1' \in \mathbb{N}^*$ and $i \geq 1$. Also by (2), $\langle\langle q_2, p\rangle\rangle$ occurs in $\hat{M}_{q_1}(s_0/u_0[u_1 \leftarrow p])$. Hence (by Lemma 4.3 applied to q_1, s_0/u_0, and u_1'), $\langle q_2, x_i\rangle$ occurs in the right-hand side of a rule of M. By (ii) of i-properness this implies that $q_2 \neq q_0$.

We now pump the tree $s_0/u_0[u_1 \leftarrow p]$ in the tree $s_0[u_0u_1 \leftarrow s] = (s_0[u_0 \leftarrow p]) \bullet (s_0/u_0[u_1 \leftarrow p]) \bullet s$: for $i \geq 0$, let $t_i = (s_0[u_0 \leftarrow p]) \bullet (s_0/u_0[u_1 \leftarrow p])^i \bullet s$. It follows from (1)–(4) that for every $i \geq 0$, $\mathrm{sts}_M(t_i, u_0u_1^i)$ contains at least one occurrence of q_1 and at least i occurrences of q_2; this is sketched in Figure 6.3 and formalized in the following claim.

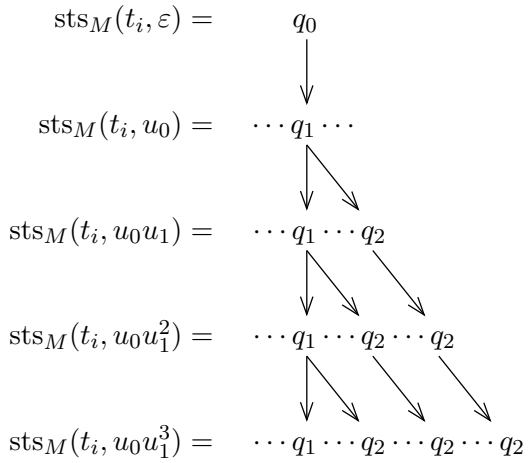


FIG. 6.3. *States that appear in state sequences of* t_i.

Claim. For all $i \geq 0$, $\#_{\langle\langle q_1,p\rangle\rangle}(\xi_i) \geq 1$ and $\#_{\langle\langle q_2,p\rangle\rangle}(\xi_i) \geq i$, where ξ_i is the tree $\hat{M}_{q_0}(t_i[u_0u_1^i \leftarrow p])$.

The proof of the claim is by induction on i. For $i = 0$, $t_i[u_0u_1^i \leftarrow p] = s_0[u_0 \leftarrow p]$ and by (1), $\#_{\langle\langle q_1,p\rangle\rangle}(\hat{M}_{q_0}(s_0[u_0 \leftarrow p])) \geq 1$. For $i+1$ we apply Lemma 4.14(i) to t_{i+1}, $u = \varepsilon$, $w = u_0u_1^i$, $v = u_0u_1^{i+1}$, and $q = q_0$. Since $h(t_{i+1}/u_0u_1^i) = h(s_0/u_0[u_1 \leftarrow s]) = \hat{h}(s_0/u_0[u_1 \leftarrow p]) = p$ by (4) and the fact that $s \in L_p$, $h(t_{i+1}/u_0u_1^{i+1}) = h(s) = p$, and $t_{i+1}[u_0u_1^i \leftarrow p] = t_i[u_0u_1^i \leftarrow p]$, we get

$$\#_{\langle\langle q',p\rangle\rangle}(\xi_{i+1}) \geq \sum_{r \in Q} \#_{\langle\langle q',p\rangle\rangle}(\hat{M}_r(s_0/u_0[u_1 \leftarrow p])) \cdot \#_{\langle\langle r,p\rangle\rangle}(\xi_i).$$

Let $q' = q_1$. Surely restricting the above sum to $r = q_1$ does not increase the result. Thus, the sum is $\geq \#_{\langle\langle q_1,p\rangle\rangle}(\hat{M}_{q_1}(s_0/u_0[u_1 \leftarrow p])) \cdot \#_{\langle\langle q_1,p\rangle\rangle}(\xi_i)$. This is ≥ 1 because $\#_{\langle\langle q_1,p\rangle\rangle}(\hat{M}_{q_1}(s_0/u_0[u_1 \leftarrow p])) \geq 1$ by (2), and $\#_{\langle\langle q_1,p\rangle\rangle}(\xi_i) \geq 1$ by induction.

Let $q' = q_2$. Now restrict the sum to $r \in \{q_1, q_2\}$. If $q_1 = q_2$, then the sum is $\geq \#_{\langle\langle q_2,p\rangle\rangle}(\hat{M}_{q_1}(s_0/u_0[u_1 \leftarrow p])) \cdot \#_{\langle\langle q_1,p\rangle\rangle}(\xi_i)$; this is $\geq 2 \cdot \max\{1, i\} \geq i + 1$, because, by (2), $\#_{\langle\langle q_2,p\rangle\rangle}(\hat{M}_{q_1}(s_0/u_0[u_1 \leftarrow p])) \geq 2$, and by induction $\#_{\langle\langle q_1,p\rangle\rangle}(\xi_i) = \#_{\langle\langle q_2,p\rangle\rangle}(\xi_i) \geq \max\{1, i\}$. If $q_1 \neq q_2$, then the sum is $\geq \#_{\langle\langle q_2,p\rangle\rangle}(\hat{M}_{q_1}(s_0/u_0[u_1 \leftarrow p])) \cdot \#_{\langle\langle q_1,p\rangle\rangle}(\xi_i) + \#_{\langle\langle q_2,p\rangle\rangle}(\hat{M}_{q_2}(s_0/u_0[u_1 \leftarrow p])) \cdot \#_{\langle\langle q_2,p\rangle\rangle}(\xi_i)$; this is $\geq i + 1$, because $\#_{\langle\langle q_2,p\rangle\rangle}(\hat{M}_{q_1}(s_0/u_0[u_1 \leftarrow p])) \geq 1$ by (2), $\#_{\langle\langle q_2,p\rangle\rangle}(\hat{M}_{q_2}(s_0/u_0[u_1 \leftarrow p])) \geq 1$ by (3), and, by induction, $\#_{\langle\langle q_1,p\rangle\rangle}(\xi_i) \geq 1$ and $\#_{\langle\langle q_2,p\rangle\rangle}(\xi_i) \geq i$. This ends the proof of the claim.

Since $\#_{\langle\langle q_2,p\rangle\rangle}(\xi_i) \geq i$, we obtain $\mathrm{size}(\tau_M(t_i)) \geq i \cdot \#_\Delta(M_{q_2}(s))$ as follows. By Lemma 4.2 and the fact that $t_i/u_0u_1^i = s$, $\tau_M(t_i) = M_{q_0}(t_i) = \xi_i[\![\ldots]\!]$ with $[\![\ldots]\!] = [\![\langle\langle q, p\rangle\rangle \leftarrow M_q(s) \mid q \in Q]\!]$. By Lemma 2.6 (summing for all $\delta \in \Delta$), $\mathrm{size}(\tau_M(t_i)) = \#_\Delta(\xi_i[\![\ldots]\!]) = S_1^\Delta + S_2^\Delta \geq S_2^\Delta = \sum_{u \in V_{\langle\langle q,p\rangle\rangle}(\xi_i), q \in Q} \#_\Delta(M_q(s)) \cdot \prod F_{\xi_i,u}^{[\![\ldots]\!]}$. Since M is nondeleting, it follows from Lemma 3.11(1) that $\#_{y_j}(M_q(s)) \geq 1$ for all $q \in Q^{(m)}$ and $j \in [m]$, and thus $\prod F_{\xi_i,u}^{[\![\ldots]\!]} \geq 1$. We get $S_2^\Delta \geq \sum_{u \in V_{\langle\langle q,p\rangle\rangle}(\xi_i), q \in Q} \#_\Delta(M_q(s)) \geq \sum_{u \in V_{\langle\langle q_2,p\rangle\rangle}(\xi_i)} \#_\Delta(M_{q_2}(s)) \geq i \cdot \#_\Delta(M_{q_2}(s))$.

Now let $s \in L_p$ such that

$$\#_\Delta(M_{q_2}(s)) > c \cdot c_1,$$

where $c_1 = \mathrm{size}(s_0/u_0[u_1 \leftarrow p]) - 1$. Then $\mathrm{size}(\tau_M(t_i)) \geq i \cdot (cc_1 + 1) = icc_1 + i$. Let $i > c(c_0 + c_2)$, where $c_0 = \mathrm{size}(s_0[u_0 \leftarrow p]) - 1$ and $c_2 = \mathrm{size}(s)$. Since $\mathrm{size}(t_i) = c_0 + ic_1 + c_2$ this means that $\mathrm{size}(\tau_M(t_i)) > c \cdot \mathrm{size}(t_i)$ because $\mathrm{size}(\tau_M(t_i)) > icc_1 + c(c_0 + c_2) = c(c_0 + ic_1 + c_2) = c \cdot \mathrm{size}(t_i)$. This contradicts $(*)$ and concludes the proof. □

We are now ready to prove step (III).

THEOREM 6.7. *Let M be a proper $MTT^R_{fnest,fcp}$. If M is lsi, then it is fci.*

Proof. If M is not fci, then, by Lemma 6.5, M is input pumpable and thus, by Lemma 6.6, M is not lsi. □

6.3. From lsi to fnest (I). In Lemma 6.6 it was proved that if a proper MTT^R M is input pumpable, then it is not lsi. So, in order to prove that M is not lsi if it is not fnest, we would like to show that if M is not fnest, then it is input pumpable. This could be done by proving a pumping argument that works on the paths of trees $\hat{M}_{q_0}(s[u \leftarrow p])$. We have chosen the following alternative: We can associate with M a top-down tree transducer A (with the same regular look-ahead as M) in such a way that

(i) the number of elements $\langle\langle q', p\rangle\rangle$ of $\langle\langle Q, \{p\}\rangle\rangle$ that appear on a path of $\hat{M}_q(s[u \leftarrow p])$ is bounded by the number of such elements that appear in $\hat{A}_q(s[u \leftarrow p])$; and

(ii) if there are n occurrences of $\langle\langle q', p\rangle\rangle$ in $\hat{A}_q(s[u \leftarrow p])$, then there are at least n occurrences of $\langle\langle q', p\rangle\rangle$ in $\hat{M}_q(s[u \leftarrow p])$.

Thus, (i) implies that if M is not fnest, then A is not fci, and (ii) implies that if A is input pumpable, then so is M. Hence we need to show that if A is not fci, then A is input pumpable. This is exactly what the application of Lemma 6.5 to A gives. (The lemma is applicable because, obviously, every top-down tree transducer is nondeleting, fnest with nesting bound 1, and fcp.)

In order to prove (i) and (ii) we merely need to require that T^R A have the same states as M (but of rank zero) and that every rule of A have the same number of occurrences of each element of $\langle Q, X\rangle$ as the corresponding rule of M.

DEFINITION 6.8 (associated T^{R}, globally fci). *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R. The T^R $A = (Q_A, P, \Sigma, \Delta, q_0, R_A, h)$ is* associated with M *if $Q_A = \{q^{(0)} \mid q \in Q\}$ and for every $q, q' \in Q$, $\sigma \in \Sigma^{(k)}$, $k \geq 0$, $i \in [k]$, and $p_1, \ldots, p_k \in P$,*

$$\#_{\langle q', x_i\rangle}(\mathrm{rhs}_A(q, \sigma, \langle p_1, \ldots, p_k\rangle)) = \#_{\langle q', x_i\rangle}(\mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)).$$

The MTT^R M is globally fci *(for short, gfci) if every T^R associated with M is fci.*

We use the subscript "gfci" for classes of translations of $\mathrm{MTT}^{\mathrm{R}}$s to denote that the corresponding transducers are gfci. Note that for T^{R}s A_1 and A_2 associated with M, $\mathrm{sts}_{A_1}(s, u)$ is a permutation of $\mathrm{sts}_{A_2}(s, u)$ (cf. Lemma 6.9 of [19]). Hence, M is gfci if and only if there *exists* a $\mathrm{T}^{\mathrm{R}}_{\mathrm{fci}}$ associated with M. For every $\mathrm{MTT}^{\mathrm{R}}$ M there is (effectively) an associated T^{R} A; it can be obtained from M by simply changing every right-hand side of M into an arbitrary right-hand side in $T_{\langle Q_A, X_k\rangle \cup \Delta}$ while preserving the number of occurrences of $\langle q, x_i\rangle$ for every $\langle q, x_i\rangle \in \langle Q, X_k\rangle$.

Let us first prove property (ii) mentioned above.

LEMMA 6.9. *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be a nondeleting MTT^R and $A = (Q_A, P, \Sigma, \Delta, q_0, R_A, h)$ a T^R associated with M. For every $q, q' \in Q$, $s \in T_\Sigma$, $u \in V(s)$, and $p \in P$, $\#_{\langle\langle q', p\rangle\rangle}(\hat{M}_q(s[u \leftarrow p])) \geq \#_{\langle\langle q', p\rangle\rangle}(\hat{A}_q(s[u \leftarrow p]))$.*

Proof. The proof is by induction on the structure of s. Let $s = \sigma(s_1, \ldots, s_k)$ with $\sigma \in \Sigma^{(k)}$ and $k \geq 0$. Let $m = \mathrm{rank}_Q(q)$.

If $u = \varepsilon$, then $\#_{\langle\langle q', p\rangle\rangle}(\hat{M}_q(s[u \leftarrow p])) = \#_{\langle\langle q', p\rangle\rangle}(\langle\langle q, p\rangle\rangle(y_1, \ldots, y_m))$, which equals (now with $q \in Q_A^{(0)}$) $\#_{\langle\langle q', p\rangle\rangle}(\langle\langle q, p\rangle\rangle) = \#_{\langle\langle q', p\rangle\rangle}(\hat{A}_q(s[u \leftarrow p]))$.

Otherwise $u = iv$ with $i \in [k]$ and $v \in V(s_i)$. Thus $\hat{M}_q(s[u \leftarrow p])$ equals $\hat{M}_q(\sigma(\tilde{s}_1, \ldots, \tilde{s}_k))$, where $\tilde{s}_\nu = s_\nu$ for $\nu \in [k] - \{i\}$ and $\tilde{s}_i = s_i[v \leftarrow p]$. For $\nu \in [k]$ let $p_\nu = \hat{h}(\tilde{s}_\nu)$. By Lemma 3.5, $\hat{M}_q(\sigma(\tilde{s}_1, \ldots, \tilde{s}_k)) = t[\![\ldots]\!]$, where $t = \mathrm{rhs}_M(q, \sigma, \langle p_1, \ldots, p_k\rangle)$ and $[\![\ldots]\!] = [\![\langle r, x_\nu\rangle \leftarrow \hat{M}_r(\tilde{s}_\nu) \mid \langle r, x_\nu\rangle \in \langle Q, X_k\rangle]\!]$. Applying Lemma 2.6 we obtain that $\#_{\langle\langle q', p\rangle\rangle}(t[\![\ldots]\!])$ equals

$$\sum_{\substack{w \in V_{\langle r, x_\nu\rangle}(t),\\ \langle r, x_\nu\rangle \in \langle Q, X_k\rangle}} \#_{\langle\langle q', p\rangle\rangle}(\hat{M}_r(\tilde{s}_\nu)) \cdot \prod F_{t,w}^{[\![\ldots]\!]}.$$

Since M is nondeleting, by Lemma 3.11(1), $\#_{y_j}(\hat{M}_r(\tilde{s}_\nu)) \geq 1$ for all $r \in Q^{(n)}$, $j \in [n]$, and $\nu \in [k]$. This implies that $\prod F_{t,w}^{[\![\ldots]\!]} \geq 1$. Hence,

$$(*) \qquad \#_{\langle\langle q', p\rangle\rangle}(\hat{M}_q(s[u \leftarrow p])) \geq \sum_{\substack{w \in V_{\langle r, x_\nu\rangle}(t),\\ \langle r, x_\nu\rangle \in \langle Q, X_k\rangle}} \#_{\langle\langle q', p\rangle\rangle}(\hat{M}_r(\tilde{s}_\nu)).$$

By induction, $\#_{\langle\langle q', p\rangle\rangle}(\hat{M}_r(\tilde{s}_i)) \geq \#_{\langle\langle q', p\rangle\rangle}(\hat{A}_r(\tilde{s}_i))$. For $\nu \in [k] - \{i\}$, $\tilde{s}_\nu \in T_\Sigma$, and therefore $\#_{\langle\langle q', p\rangle\rangle}(\hat{M}_r(\tilde{s}_\nu)) = \#_{\langle\langle q', p\rangle\rangle}(M_r(\tilde{s}_\nu)) = 0 = \#_{\langle\langle q', p\rangle\rangle}(A_r(\tilde{s}_\nu)) = \#_{\langle\langle q', p\rangle\rangle}(\hat{A}_r(\tilde{s}_\nu))$. Thus, the sum in $(*)$ is $\geq \sum_{w \in V_{\langle r, x_\nu\rangle}(t), \langle r, x_\nu\rangle \in \langle Q, X_k\rangle} \#_{\langle\langle q', p\rangle\rangle}(\hat{A}_r(\tilde{s}_\nu))$. Since A is associated with M, $|V_{\langle r, x_\nu\rangle}(\zeta)| = |V_{\langle r, x_\nu\rangle}(t)|$ for every $\langle r, x_\nu\rangle \in \langle Q, X_k\rangle$, where $\zeta = \mathrm{rhs}_A(q, \sigma, \langle p_1, \ldots, p_k\rangle)$. Therefore the above sum does not change if we replace t by ζ. Then by Lemma 2.4 we get $\#_{\langle\langle q', p\rangle\rangle}(\zeta[\ldots])$ with $[\ldots] = [\langle r, x_\nu\rangle \leftarrow \hat{A}_r(\tilde{s}_\nu) \mid \langle r, x_\nu\rangle \in \langle Q_A, X_k\rangle]$. By Lemma 3.5 and the fact that $\hat{A}$ is a T^{R}, this equals $\#_{\langle\langle q', p\rangle\rangle}(\hat{A}_q(s[u \leftarrow p]))$. □

For a nondeleting $\mathrm{MTT}^{\mathrm{R}}$ M it follows immediately from Lemma 6.9 and Definition 6.4 that if a T^{R} A associated with M is input pumpable, then M is also input pumpable.

LEMMA 6.10. *Let M be a nondeleting MTT^R and A a T^R associated with M. If A is input pumpable, then so is M.*

From Lemma 6.9 it also follows that gfci is a generalization of fci: if $\#_{\langle\langle Q,\{p\}\rangle\rangle}(\hat{M}_{q_0}(s[u \leftarrow p]))$ is bounded by some N, then so is $\#_{\langle\langle Q,\{p\}\rangle\rangle}(\hat{A}_{q_0}(s[u \leftarrow p]))$; i.e., if M is fci, then it is gfci. However, the converse is not true: there are $\mathrm{MTT^R}$s which are gfci but not fci. In fact, even for fcp $\mathrm{MTT^R}$s, gfci does *not* imply fci. To see this consider an MTT M which contains the following rules (and trivial look-ahead $P = \{p\}$):

$$\begin{array}{lcl} \langle q_0, \sigma(x_1, x_2)\rangle & \to & \langle q, x_1\rangle(\langle q_0, x_2\rangle), \\ \langle q_0, \alpha\rangle & \to & \alpha, \\ \langle q, \sigma(x_1, x_2)\rangle(y_1) & \to & \sigma(y_1, y_1), \\ \langle q, \alpha\rangle(y_1) & \to & \sigma(y_1, y_1). \end{array}$$

Now let $s_0 = \alpha$ and for $n \geq 0$ let $s_{n+1} = \sigma(\alpha, s_n)$. Then

$$\begin{aligned} \langle q_0, s_n\rangle &\Rightarrow_M \langle q, \alpha\rangle(\langle q_0, s_{n-1}\rangle) \\ &\Rightarrow_M \sigma(\langle q_0, s_{n-1}\rangle, \langle q_0, s_{n-1}\rangle) \\ &\Rightarrow^*_M \sigma(\sigma(\langle q_0, s_{n-2}\rangle, \langle q_0, s_{n-2}\rangle), \sigma(\langle q_0, s_{n-2}\rangle, \langle q_0, s_{n-2}\rangle)). \end{aligned}$$

Hence, $\hat{M}_{q_0}(s_n[2^n \leftarrow p])$ is a full binary tree of height n with all leaves labeled $\langle\langle q_0, p\rangle\rangle$. Thus $\mathrm{sts}_M(s_n, 2^n) = q_0^{2^n}$, which means that M is not fci. However, M is gfci and fcp, with bounds 1 and 2, respectively. To see that M is gfci, consider the $\mathrm{T^R}$ A with right-hand side $\sigma(\langle q, x_1\rangle, \langle q_0, x_2\rangle)$ for the (q_0, σ)-rule and right-hand side α for all other rules. Now A is associated with M, and it is linear in the input variables x_i; i.e., A is fci with bound 1. Moreover, M is not lsi (because $\tau_M(s_n)$ is a full binary tree of height n). Thus, gfci plus fcp cannot be taken as an alternative to the definition of finite copying: $MTT^R_{\mathrm{fci,fcp}} \subsetneq MTT^R_{\mathrm{gfci,fcp}}$.

As illustrated by the example above, a gfci $\mathrm{MTT^R}$ M need not be fci, and thus the number of occurrences of elements of $\langle\langle Q, \{p\}\rangle\rangle$ in $\hat{M}_{q_0}(s[u \leftarrow p])$ is in general unbounded due to parameter copying (in the example above by the rules with right-hand side $\sigma(y_1, y_1)$). However, the number of such elements that appear on *one path* in $\hat{M}_{q_0}(s[u \leftarrow p])$ is bounded, and thus M is fnest. To see this intuitively, consider a label path π in a tree in $T_{\langle Q, T_\Sigma\rangle \cup \Delta}$. The application of a rule r of an $\mathrm{MTT^R}$ does not copy any states on the path π; thus, it increases the number of occurrences of q' on π by at most $\#_{\langle\{q'\},X\rangle}(\mathrm{rhs}(r))$, which equals $\#_{\langle\{q'\},X\rangle}(\mathrm{rhs}(r'))$ for the corresponding rule r' of a $\mathrm{T^R}$ associated with M. We now give a formal proof of property (i) mentioned above.

LEMMA 6.11. *Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT^R and $A = (Q_A, P, \Sigma, \Delta, q_0, R_A, h)$ a T^R associated with M. For every $q, q' \in Q$, $s \in T_\Sigma$, $u \in V(s)$, $p \in P$, and every label path π in $\hat{M}_q(s[u \leftarrow p])$, $\#_{\langle\langle q', p\rangle\rangle}(\pi) \leq \#_{\langle\langle q', p\rangle\rangle}(\hat{A}_q(s[u \leftarrow p]))$.*

Proof. The proof is by induction on the length of u.

For $u = \varepsilon$, $\#_{\langle\langle q', p\rangle\rangle}(\pi) = \#_{\langle\langle q', p\rangle\rangle}(\langle\langle q, p\rangle\rangle) = \#_{\langle\langle q', p\rangle\rangle}(\hat{A}_q(s[u \leftarrow p]))$.

For $u = u'i$ it follows from Lemma 4.3 that $\hat{M}_q(s[u \leftarrow p]) = t[\![i]\!][\![..]\!]$ with $t = \hat{M}_q(s[u' \leftarrow p'])[\![\mathrm{rhs}]\!]$, $p' = \hat{h}(s/u'[i \leftarrow p])$, and the substitutions $[\![\mathrm{rhs}]\!]$, $[\![..]\!]$, and $[\![i]\!]$ defined as in Lemma 4.3 (with u' instead of u, p' instead of p, and p instead of p_i). By Lemma 2.3(i) applied to $t'[\![..]\!]$ with $t' = t[\![i]\!]$, the label path π is of the form $w_0 v_1 w_1 \cdots v_m w_m$, $m \geq 0$, where $\pi' = w_0\langle r_1, x_{\nu_1}\rangle w_1 \cdots \langle r_m, x_{\nu_m}\rangle w_m$ is a label path in t', and for $j \in [m]$, $r_j \in Q$, $\nu_j \in [k] - \{i\}$, v_j is a label path in $M_{r_j}(s/u'\nu_j)$, and

$w_0, \dots, w_m$ do not contain elements of $\langle Q, X_k - \{x_i\}\rangle$. Since $M_{r_j}(s/u'\nu_j) \in T_\Delta(Y)$, $\#_{\langle\langle q',p\rangle\rangle}(v_j) = 0$ for all $j \in [m]$, which means that $\#_{\langle\langle q',p\rangle\rangle}(\pi) = \#_{\langle\langle q',p\rangle\rangle}(\pi')$.

Clearly, by the definition of $[\![i]\!]$, $\#_{\langle\langle q',p\rangle\rangle}(\pi') = \#_{\langle q',x_i\rangle}(\pi'')$ for some label path π'' in t. Hence, it remains to show that $\#_{\langle q',x_i\rangle}(\pi'') \le \#_{\langle\langle q',p\rangle\rangle}(\hat{A}_q(s[u \leftarrow p])) = \#_{\langle\langle q',p\rangle\rangle}(\xi[\mathrm{rhs}][..][i]) = \#_{\langle q',x_i\rangle}(\xi[\mathrm{rhs}])$, where $\xi = \hat{A}_q(s[u' \leftarrow p'])$ and [rhs], [..], [i] are the (corresponding first-order variants of the) substitutions of Lemma 4.3.

By Lemma 2.3(i) applied to the tree $t = \hat{M}_q(s[u' \leftarrow p'])[\![\mathrm{rhs}]\!]$, π'' is of the form $w_0v_1w_1\cdots v_mw_m$, $m \ge 0$, where $\rho = w_0\langle\langle r_1, p'\rangle\rangle w_1 \cdots \langle\langle r_m, p'\rangle\rangle w_m$ is a label path in $\hat{M}_q(s[u' \leftarrow p'])$ and for $j \in [m]$, $r_j \in Q$, v_j is a label path in $\mathrm{rhs}_M(r_j, \sigma, \langle p_1, \dots, p_k\rangle)$, and $w_0, \dots, w_m$ contain no elements of $\langle\langle Q, \{p'\}\rangle\rangle$ (i.e., w_j is a string over $\Delta \cup Y$). Thus, $\#_{\langle q',x_i\rangle}(\pi'') = \sum_{j\in[m]} \#_{\langle q',x_i\rangle}(v_j)$. Since, for $j \in [m]$, v_j is a label path in $\mathrm{rhs}_M(r_j, \sigma, \langle p_1, \dots, p_k\rangle)$, this sum is surely

$$\le \sum_{j\in[m]} \#_{\langle q',x_i\rangle}(\mathrm{rhs}_M(r_j, \sigma, \langle p_1, \dots, p_k\rangle)) = \sum_{j\in[m]} \#_{\langle q',x_i\rangle}(\mathrm{rhs}_A(r_j, \sigma, \langle p_1, \dots, p_k\rangle)),$$

which can be written as

$$\sum_{r\in Q} \#_{\langle\langle r,p'\rangle\rangle}(\rho) \cdot \#_{\langle q',x_i\rangle}(\mathrm{rhs}_A(r, \sigma, \langle p_1, \dots, p_k\rangle)).$$

By induction this is $\le \sum_{r\in Q} \#_{\langle\langle r,p'\rangle\rangle}(\xi) \cdot \#_{\langle q',x_i\rangle}(\mathrm{rhs}_A(r, \sigma, \langle p_1, \dots, p_k\rangle))$, which equals $\#_{\langle q',x_i\rangle}(\xi[\mathrm{rhs}])$ by Lemma 2.4. □

It follows immediately from Lemma 6.11, by taking $q = q_0$ and summing over all $q' \in Q$, that if A is fci, then M is fnest, with the same bound. This is stated in the next lemma.

LEMMA 6.12. *If an MTT^R is gfci, then it is fnest.*

We are now ready to prove step (I), i.e., that for a proper MTT$^{\mathrm{R}}$, lsi implies fnest.

THEOREM 6.13. *Let M be a proper MTT^R. If M is lsi, then it is fnest.*

Proof. If M is not fnest, then by Lemma 6.12 it is not gfci. By the definition of gfci this means that any T$^{\mathrm{R}}$ A associated with M is not fci. The application of Lemma 6.5 to A gives that A is input pumpable, and thus by Lemma 6.10 M is input pumpable. Now Lemma 6.6 implies that M is not lsi. □

From Theorems 6.13, 6.3, and 6.7 we obtain the main result of this section: the converse of Theorem 4.19 for proper MTT$^{\mathrm{R}}$s.

THEOREM 6.14. *Let M be a proper MTT^R. If M is lsi, then it is finite copying.*

Recall from section 4.3 the notion of finite contribution. By Lemma 4.18, every finite copying MTT$^{\mathrm{R}}$ is of finite contribution, and by the discussion before Theorem 4.19, every MTT$^{\mathrm{R}}$ of finite contribution is lsi. Together with Theorem 6.14 this shows that a proper MTT$^{\mathrm{R}}$ is finite copying if and only if it is of finite contribution. It can be proved that this even holds for a productive MTT$^{\mathrm{R}}$ that satisfies (ii) of Definition 5.6 (of p-properness). Thus, the notions of finite copying and finite contribution are closely related.

7. Main results and consequences. In this final section we prove our main results: (i) a translation is MSO definable if and only if it is a macro tree translation of linear size increase, and (ii) for a given MTT M it is decidable whether or not τ_M is MSO definable. Then we discuss some consequences of these results for top-down tree transducers, attributed tree transducers, and context-free graph grammars. At last some open problems and further research topics are mentioned.

THEOREM 7.1. *Let M be an MTT^R. Then the following statements are equivalent:*

(1) *τ_M is MSO definable.*

(2) *τ_M is lsi.*

(3) *prop(M) is finite copying.*

Proof. Since every MSO definable tree translation is lsi (see section 2.5), (1) $\Rightarrow$ (2). Note that this can also be proved using the results from section 4: If τ_M is MSO definable, then by Lemma 4.9, $\tau_M \in MTT^R_{\text{fc}}$ and thus, by Theorem 4.19, τ_M is lsi. To show (2) $\Rightarrow$ (3), let τ_M be lsi. By Theorem 5.9, there is a proper MTT$^{\text{R}}$ prop(M) with $\tau_{\text{prop}(M)} = \tau_M$; i.e., $\tau_{\text{prop}(M)}$ is lsi. By Theorem 6.14, prop(M) is finite copying. Finally, if prop(M) is finite copying, then, by Lemma 4.9, $\tau_M = \tau_{\text{prop}(M)}$ is MSO definable. Thus (3) $\Rightarrow$ (1). □

Note that, as discussed at the end of section 6, we could have included "(4) prop(M) is of finite contribution" as another equivalent statement in Theorem 7.1.

Theorem 7.1 shows that the class *MSOTT* of MSO definable tree translations can be characterized as those macro tree translations that are lsi. Recall (from section 2.5) that *LSI* denotes the class of all lsi tree translations.

THEOREM 7.2. *$MSOTT = MTT \cap LSI$.*

Proof. If $\tau \in MTT \cap LSI$, then there is an MTT M such that $\tau_M = \tau$ is lsi. By Theorem 7.1 τ_M is MSO definable, and thus $MTT \cap LSI \subseteq MSOTT$. If $\tau \in MSOTT$, then by Lemma 4.9 there is an MTT$^{\text{R}}$ M with $\tau_M = \tau$. By Theorem 7.1 τ_M is lsi, and thus $MSOTT \subseteq MTT^R \cap LSI$. By Lemma 3.4, $MTT^R = MTT$. □

By Theorem 7.1, the proper normal form prop(M) (which can be constructed by Theorem 5.9) of an MTT M is finite copying if and only if τ_M is MSO definable. Since the finite copying property is decidable (Lemma 4.10) this implies that for M it is decidable whether or not τ_M is MSO definable. If prop(M) is finite copying, then an MSO tree transducer that realizes τ_M can be constructed, because the equality $MSOTT = MTT^R_{\text{fc}}$ of Lemma 4.9 is effective (cf. the discussion following Lemma 4.10).

THEOREM 7.3. *It is decidable for an MTT M whether or not τ_M is MSO definable, and if it is, then an MSO tree transducer for τ_M can be constructed.*

7.1. Top-down tree transducers. A top-down tree transducer can translate a monadic tree (of height n) into a full binary tree (of height n). This translation is of exponential size increase, and hence it is not MSO definable. On the other hand, there are MSO definable tree translations that cannot be realized by top-down tree transducers: consider the translation that associates with a tree its yield (i.e., the left-to-right sequence of the labels of its leaves), seen as a monadic tree. This translation is MSO definable (cf. Example 1(6, yield) of [3]) but it cannot be realized by a top-down tree transducer, because it is of exponential height increase (viz. it translates a full binary tree of height n into its yield, a monadic tree of height 2^n), whereas top-down tree translations are of linear height increase (cf. Lemma 3.27 of [28]). Now, which translations realized by top-down tree transducers (with regular look-ahead) are MSO definable? By our results, they are exactly the translations realized by finite copying T$^{\text{R}}$s.

THEOREM 7.4. *$T^R \cap MSOTT = T^R_{fc}$.*

Proof. Let M be a T$^{\text{R}}$ such that τ_M is MSO definable. By Theorem 7.1, prop(M) is finite copying. By Theorem 5.9, prop(M) is a T$^{\text{R}}$. Thus, $\tau_M = \tau_{\text{prop}(M)} \in T^R_{\text{fc}}$. Hence, $T^R \cap MSOTT \subseteq T^R_{\text{fc}}$. The inclusion $T^R_{\text{fc}} \subseteq T^R \cap MSOTT$ is immediate from Lemma 4.9. □

It is shown in Theorem 7.4 of [19] that $T^R_{\mathrm{fc}} = MSOTT_{\mathrm{dir}}$: the so-called direction preserving MSO definable tree translations. An MSO tree transducer (see section 2.5) is direction preserving if $s \models \chi_{i,c,d}(x,y)$ implies that y is a descendant of x in the input tree s. Thus, the descendant relation in the output tree can only hold between nodes that are also (copies of) descendants in the input tree.

Note that it follows immediately from Theorem 7.1 that $T^R \cap MSOTT = T^R \cap LSI$. Thus, $T^R_{\mathrm{fc}} = T^R \cap LSI$. Since $\mathrm{T}^{\mathrm{R}}_{\mathrm{fc}}$s are closely related to tree-walking transducers (see Theorem 4.9 of [22]), this may be viewed as the result of [1] that the translations realized by tree-walking transducers are exactly the generalized syntax-directed translations of linear size increase.

7.2. Attributed tree transducers. Attributed tree transducers [27, 28] serve as a formal model for attribute grammars [37]. As argued in [3], adding the feature of look-ahead to them yields a better model of attribute grammars and a more robust class of tree translations. Let ATT^R denote the class of translations realized by attributed tree transducers with look-ahead (see [3, 19]) and let the subscript "sur" denote that the transducers are "single-use restricted" (cf. section 5 in [19]); i.e., for every input symbol σ, each outside attribute is used at most once in the set of rules for σ. It is proved in Theorem 17 of [3] that $MSOTT = ATT^R_{\mathrm{sur}}$. Hence $MSOTT \subseteq ATT^R \cap LSI$. Equality of these classes now follows from Theorem 7.2 and the fact that $ATT^R \subseteq MTT$. (The latter inclusion can be proved as follows: By definition, ATT^R consists of all translations that can be realized by the composition of an attributed relabeling, followed by an attributed tree translation. It follows from Theorem 4.4 of [19] that attributed relabelings can be realized by T^{R}s. Thus, $ATT^R \subseteq T^R \circ ATT$, where ATT denotes the class of translations realized by attributed tree transducers. By Lemma 5.11 of [19], $ATT \subseteq MTT^R$ and so $T^R \circ ATT \subseteq T^R \circ MTT^R$, which, by Lemma 3.4, equals $T^R \circ MTT$. Since regular look-ahead can be realized by first running a finite state relabeling, i.e., applying a translation in $DBQREL$ (cf. Theorem 2.6 of [14]), we get the inclusion in $DBQREL \circ T \circ MTT$, which is $\subseteq DBQREL \circ MTT$ by Corollary 4.10 of [24], and thus we have the inclusion in $MTT^R = MTT$.)

THEOREM 7.5. *$MSOTT = ATT^R \cap LSI$.*

From the fact that $ATT^R \subseteq MTT$ (effectively) together with Theorem 7.3 and the fact that $MSOTT = ATT^R_{\mathrm{sur}}$ (effectively), we obtain the following decidability result for attributed tree transducers.

THEOREM 7.6. *For an ATT^R A it is decidable whether or not there exists an equivalent single-use restricted ATT^R A', and if so, A' can be constructed.*

The interpretation of Theorem 7.6 in terms of classical attribute grammars involves a technical detail: roughly speaking, the look-ahead part of an $\mathrm{ATT}^{\mathrm{R}}$ corresponds to the underlying context-free grammar of an attribute grammar. If we want to apply Theorem 7.6 to an attribute grammar G, then we first have to turn G into an equivalent $\mathrm{ATT}^{\mathrm{R}}$ A, i.e., into an $\mathrm{ATT}^{\mathrm{R}}$ that realizes the same tree-to-tree translation as G (translating the non-derivation-trees of G into some error symbol). Now assume that for A there is an equivalent single-use restricted $\mathrm{ATT}^{\mathrm{R}}$ A'. In general the look-ahead of A' will be different from the one of A, which implies that an attribute grammar G' equivalent to A' does *not* have the same underlying context-free grammar as G, and hence the tree-to-tree translation realized by G' is different from the one realized by G. This problem can be avoided by adding boolean-valued attributes to G' (cf. the introduction of [3]), which simulate the look-ahead part of A'. In this way G' and G have the same underlying context-free grammar and they realize the

same tree-to-tree translation (however, the boolean-valued attributes are, in general, not single-use restricted).

7.3. Context-free graph grammars. A context-free graph grammar (see, e.g., [16]) generates a graph language. If the graphs are restricted to trees, then we obtain a tree language. As discussed in the introduction of [19], the class of tree languages that can be generated by context-free graph grammars (either by hyperedge replacement (HR), or by node replacement (NR); cf. Section 6 of [16]) can be obtained by applying the MSO definable tree translations to the regular tree languages. By Theorem 7.2 it means that this class of tree languages can be obtained by the application of lsi macro tree translations to the regular tree languages. This is just a straightforward variation of similar statements in the literature: for single-use restricted ATTs in Corollary 19 of [3], for "single-use restricted" MTTs and for finite copying MTTs in Corollary 7.3 of [19], and for nondeleting MTTs that are finite copying and linear in the parameters in Theorem 5 of [20] (based on Theorem 8.1 of [10]).

THEOREM 7.7. *The output tree languages of MTTs of linear size increase applied to the regular tree languages are the tree languages generated by (HR or NR) context-free graph grammars.*

7.4. Open problems and further research topics. We have proved that for an MTT it is decidable whether or not the translation it realizes is MSO definable. What is the complexity of this problem? In fact, the complexity of deciding the finiteness of ranges of (compositions of) MTTs [12] (cf. Lemma 3.8) is not known, and our decidability proof is based on this result. Generally speaking, complexity issues have not yet been studied in the area of MTTs. A first result in that direction is [42], which shows that for a composition τ of deterministic macro tree translations and an input tree s, the corresponding output tree $\tau(s)$ can be computed in time linear in the sum of the sizes of s and t.

Is it decidable for an MSO tree transducer whether its translation is in T^R, i.e., whether it can be simulated by a top-down tree transducer with regular look-ahead (see section 7.1)?

It would be interesting to find a classification of the possible size increases of MTTs. For top-down tree transducers such a classification is given in [1], and it is shown that the size increase of every top-down tree transducer is either polynomial or exponential. For MTTs it could be the case that every size increase is either polynomial, exponential, or double exponential.

Is polynomial size increase decidable for MTTs? If so, what is the complexity? For top-down tree transducers it is shown in [11] that this problem is NLOGSPACE-complete. It is not clear how MSO definability could be generalized in order to obtain the class of polynomial size increase macro tree translations. (Note that there are well-established models of polynomial size increase transducers based on first-order logic; see, e.g., [13, 34].)

Composition of MTTs yields a proper hierarchy; i.e., there are translations which can be realized by the composition of $m + 1$ MTTs but not by the composition of m MTTs (Theorem 4.16 of [24]). Now, what happens if we restrict our attention to translations that are of linear size increase? Maybe then composition does *not* yield a proper hierarchy, but rather it remains the class of MSO definable tree translations, i.e., is $LSI \cap \bigcup_n MTT^n = MSOTT$. Since compositions of MTTs can be realized by high-level tree transducers (and vice versa) [25] this question is equivalent to the following: Are lsi high-level tree translations MSO definable? Again, this question could also be considered for polynomial instead of linear size increase.

Recently we have shown that the k-pebble tree transducer, introduced in [45] as a formal model for XML query languages, can be simulated by the composition of MTTs [21]. Thus, if, as discussed in the previous paragraph, our results would even hold for compositions of MTTs, then they would also hold for k-pebble tree transducers.

For both MTTs and MSO transducers there are nondeterministic variants (cf. [24] and [5], respectively). We would like to know whether our result carries over to the nondeterministic case, i.e., whether the nondeterministic macro tree translations of linear size increase are precisely the nondeterministic MSO definable tree translations.

Last but not least: Given an MTT M, is it decidable whether the translation τ_M realized by M can be realized by an attributed tree transducer (with look-ahead), i.e., is it decidable whether $\tau_M \in ATT$ (or ATT^R)? This question is of interest because, as shown in [3], ATT^R is the class of tree translations that are MSO definable (by an MSO tree-to-graph transducer; see [5]) when subtrees can be shared in the output tree. Of course, if τ_M is MSO definable (without sharing of subtrees), which can be decided by Theorem 7.3, then the answer to the above question is positive, because $MSOTT = ATT^R_{\mathrm{sur}}$ by the result of [3] (other positive criteria are discussed in [8, 9, 29]). On the other hand, note that $\mathrm{ATT^R}$s are of linear size-to-height increase (cf., e.g., Lemma 5.40 of [28]). Denote by *LSHI* the class of all translations of linear size-to-height increase. Probably it can be proved (by methods similar to those in this paper) that $MTT \cap LSHI = MTT^R_{\mathrm{fnest}}$ and that $\tau_M \in LSHI$ if and only if $\mathrm{prop}(M)$ is fnest, which is decidable. Thus, it would be decidable for an MTT whether or not it is of linear size-to-height increase. If it is not, then it cannot be realized by an $\mathrm{ATT^R}$. But these are only partial answers.

Acknowledgment. We thank the referees for their constructive comments.

REFERENCES

[1] A. V. Aho and J. D. Ullman, *Translations on a context-free grammar*, Inform. and Control, 19 (1971), pp. 439–475.
[2] G. J. Bex, S. Maneth, and F. Neven, *A formal model for an expressive fragment of XSLT*, Inform. Systems, 27 (2002), pp. 21–39.
[3] R. Bloem and J. Engelfriet, *A comparison of tree transductions defined by monadic second order logic and by attribute grammars*, J. Comput. System Sci., 61 (2000), pp. 1–50.
[4] B. Courcelle, *Fundamental properties of infinite trees*, Theoret. Comput. Sci., 25 (1983), pp. 95–169.
[5] B. Courcelle, *Monadic second-order definable graph transductions: A survey*, Theoret. Comput. Sci., 126 (1994), pp. 53–75.
[6] B. Courcelle, *Structural properties of context-free sets of graphs generated by vertex replacement*, Inform. and Comput., 116 (1995), pp. 275–293.
[7] B. Courcelle and J. Engelfriet, *A logical characterization of the sets of hypergraphs defined by hyperedge replacement grammars*, Math. Systems Theory, 28 (1995), pp. 515–552.
[8] B. Courcelle and P. Franchi-Zannettacci, *Attribute grammars and recursive program schemes.* I, Theoret. Comput. Sci., 17 (1982), pp. 163–191.
[9] B. Courcelle and P. Franchi-Zannettacci, *Attribute grammars and recursive program schemes.* II, Theoret. Comput. Sci., 17 (1982), pp. 235–257.
[10] F. Drewes, *A characterization of the sets of hypertrees generated by hyperedge-replacement graph grammars*, Theory Comput. Systems, 32 (1999), pp. 159–208.
[11] F. Drewes, *The complexity of the exponential output size problem for top-down and bottom-up tree transducers*, Inform. and Comput., 169 (2001), pp. 264–283.
[12] F. Drewes and J. Engelfriet, *Decidability of finiteness of ranges of tree transductions*, Inform. and Comput., 145 (1998), pp. 1–50.
[13] H.-D. Ebbinghaus and J. Flum, *Finite Model Theory*, Springer-Verlag, Berlin, 1995.

[14] J. Engelfriet, *Top-down tree transducers with regular look-ahead*, Math. Systems Theory, 10 (1977), pp. 289–303.
[15] J. Engelfriet, *Some open questions and recent results on tree transducers and tree languages*, in Formal Language Theory: Perspectives and Open Problems, R. Book, ed., Academic Press, New York, 1980, pp. 241–286.
[16] J. Engelfriet, *Context-free graph grammars*, in Handbook of Formal Languages, Volume 3, G. Rozenberg and A. Salomaa, eds., Springer-Verlag, Berlin, 1997, pp. 125–213.
[17] J. Engelfriet and G. Filè, *The formal power of one-visit attribute grammars*, Acta Inform., 16 (1981), pp. 275–302.
[18] J. Engelfriet and H. J. Hoogeboom, *MSO definable string transductions and two-way finite-state transducers*, ACM Trans. Comput. Log., 2 (2001), pp. 216–254.
[19] J. Engelfriet and S. Maneth, *Macro tree transducers, attribute grammars, and MSO definable tree translations*, Inform. and Comput., 154 (1999), pp. 34–91.
[20] J. Engelfriet and S. Maneth, *Tree languages generated by context-free graph grammars*, in Proceedings of the 6th International Workshop on Theory and Application of Graph Transformations (TAGT'98), H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, eds., Lecture Notes in Comput. Sci. 1764, Springer-Verlag, New York, 2000, pp. 15–29.
[21] J. Engelfriet and S. Maneth, *A comparison of pebble tree transducers with macro tree transducers*, Acta Inform., to appear.
[22] J. Engelfriet, G. Rozenberg, and G. Slutzki, *Tree transducers, L systems, and two-way machines*, J. Comput. System Sci., 20 (1980), pp. 150–202.
[23] J. Engelfriet and V. van Oostrom, *Logical description of context-free graph languages*, J. Comput. System Sci., 55 (1997), pp. 489–503.
[24] J. Engelfriet and H. Vogler, *Macro tree transducers*, J. Comput. System Sci., 31 (1985), pp. 71–146.
[25] J. Engelfriet and H. Vogler, *High level tree transducers and iterated pushdown tree transducers*, Acta Inform., 26 (1988), pp. 131–192.
[26] J. Engelfriet and H. Vogler, *The translation power of top-down tree-to-graph transducers*, J. Comput. System Sci., 49 (1994), pp. 258–305.
[27] Z. Fülöp, *On attributed tree transducers*, Acta Cybernet., 5 (1981), pp. 261–279.
[28] Z. Fülöp and H. Vogler, *Syntax-Directed Semantics—Formal Models based on Tree Transducers*, Monogr. Theoret. Comput. Sci. EATCS Ser., W. Brauer, G. Rozenberg, and A. Salomaa, eds., Springer-Verlag, Berlin, 1998.
[29] Z. Fülöp and H. Vogler, *A characterization of attributed tree transformations by a subclass of macro tree transducers*, Theory Comput. Systems, 32 (1999), pp. 649–676.
[30] H. Ganzinger, *Increasing modularity and language-independency in automatically generated compilers*, Sci. Comput. Programming, 3 (1983), pp. 223–278.
[31] F. Gécseg and M. Steinby, *Tree Automata*, Akadémiai Kiadó, Budapest, 1984.
[32] F. Gécseg and M. Steinby, *Tree automata*, in Handbook of Formal Languages, Volume 3, G. Rozenberg and A. Salomaa, eds., Springer-Verlag, Berlin, 1997, pp. 1–68.
[33] R. Giegerich, *Composition and evaluation of attribute coupled grammars*, Acta Inform., 25 (1988), pp. 355–423.
[34] N. Immerman, *Descriptive Complexity*, Springer-Verlag, New York, 1999.
[35] E. Irons, *A syntax directed compiler for ALGOL* 60, Comm. ACM, 4 (1961), pp. 51–55.
[36] N. Klarlund and M. I. Schwartzbach, *Graphs and decidable transductions based on edge constraints*, in Proceedings of the 19th Colloquium on Trees in Algebra and Programming—CAAP 94, S. Tison, ed., Lecture Notes in Comput. Sci. 787, Springer-Verlag, New York, 1994, pp. 187–201.
[37] D. Knuth, *Semantics of context-free languages*, Math. Systems Theory, 2 (1968), pp. 127–145. (Corrections in Math. Systems Theory, 5 (1971), pp. 95–96.)
[38] H.-P. Kolb, J. Michaelis, U. Mönnich, and F. Morawietz, *An operational and denotational approach to non-context-freeness*, Theoret. Comput. Sci., to appear.
[39] A. Kühnemann, *Benefits of tree transducers for optimizing functional programs*, in Proceedings of the 18th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'98), V. Arvind and R. Ramanujam, eds., Lecture Notes in Comput. Sci. 1530, Springer-Verlag, New York, 1998, pp. 146–157.
[40] A. Kühnemann and H. Vogler, *Attributgrammatiken*, Vieweg-Verlag, Braunschweig, Wiesbaden, Germany, 1997.
[41] A. Kühnemann and J. Voigtländer, *Tree Transducer Composition as Deforestation Method for Functional Programs*, Tech. Report TUD-FI01-07, Department of Computer Science, Technical University of Dresden, 2001.
[42] S. Maneth, *The complexity of compositions of deterministic tree transducers*, in Proceedings

of the 22nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2002), M. Agrawal and A. Seth, eds., Lecture Notes in Comput. Sci. 2556, Springer-Verlag, New York, 2002, pp. 265–276.

[43] S. Maneth and F. Neven, *Recursive structured document transformations*, in Research Issues in Structured and Semistructured Database Programming—Revised Papers DBPL'99, R. Connor and A. Mendelzon, eds., Lecture Notes in Comput. Sci. 1949, Springer-Verlag, New York, 2000, pp. 80–98.

[44] J. Michaelis, U. Mönnich, and F. Morawietz, *On minimalist attribute grammars and macro tree transducers*, in Linguistic Form and Its Computation, C. Rohrer, A. Rossdeutscher, and H. Kamp, eds., CSLI Publications, Stanford, CA, 2001, pp. 287–326.

[45] T. Milo, D. Suciu, and V. Vianu, *Typechecking for XML transformers*, in Proceedings of the 19th ACM Symposium on Principles of Database Systems (PODS'2000), ACM, New York, 2000, pp. 11–22.

[46] W. Rounds, *Mappings and grammars on trees*, Math. Systems Theory, 4 (1970), pp. 257–287.

[47] J. Stoy, *Denotational Semantics*, MIT Press, Cambridge, MA, 1977.

[48] J. Thatcher, *Generalized^2 sequential machine maps*, J. Comput. System Sci., 4 (1970), pp. 339–367.

[49] A. van Deursen, J. Heering, and P. Klint, eds., *Language Prototyping*, AMAST Series in Computing 5, World Scientific, River Edge, NJ, 1996.

[50] A. van Deursen, P. Klint, and F. Tip, *Origin tracking and its applications*, in Language Prototyping, World Scientific, River Edge, NJ, 1996, pp. 249–294.

[51] V. Vianu, *A web odyssey: From Codd to XML*, in Proceedings of the 20th ACM Symposium on Principles of Database Systems (PODS'2001), ACM, New York, 2001, pp. 1–15.

[52] H. Vogler, *Functional description of the contextual analysis in block-structured programming languages: A case study of tree transducers*, Sci. Comput. Programming, 16 (1991), pp. 251–275.

[53] R. Wilhelm and D. Maurer, *Compiler Design*, Addison-Wesley, Reading, MA, 1995.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 1007–1039

THE IMPACT OF TIMING KNOWLEDGE ON THE SESSION PROBLEM*

INJONG RHEE† AND JENNIFER L. WELCH‡

Abstract. The *session* problem is an abstraction of fundamental synchronization problems in distributed systems. It has previously been used as a test case to demonstrate the differences in the time needed to solve problems in several timing models.

The goal of this paper is to compare the computational power of a family of partially synchronous models by studying the time needed to solve the session problem. Four timing parameters are considered: the maximum and minimum process step times and message delays. Timing models are obtained by considering independently whether each parameter is known (i.e., is hard-wired into the processes' code) or unknown, giving rise to four shared memory models and 16 message passing models. The models are compared based on the time complexity, measured in real time, of the session problem.

This paper presents a modular proof technique for obtaining asymptotically tight bounds on the time complexity of the session problem for the four shared memory models and the 16 message passing models. Timing information known in each particular model can be exploited by algorithms to count sessions in different ways. This paper reports five different counting algorithms. The matching lower bound for each model suggests that they are the optimal ways to count sessions. Based on these bounds, a lattice among unknown parameter models is constructed, which confirms the common belief that as more timing information is known in a model, the model behaves more like a synchronous system.

Key words. session problem, partially synchronous models, timing information, distributed computing, upper bounds, lower bounds

AMS subject classifications. 68Q17, 68W15

DOI. 10.1137/S009753979936094X

1. Introduction.

Early work in distributed computing usually assumed one of two extreme timing models: the completely synchronous model, in which processes operate in lockstep rounds of computation and a message sent in a round is delivered in the next round, or the completely asynchronous model, in which there are no bounds on process step time or message delay. However, in most distributed systems, processes operate neither in lockstep nor at completely independent rates. Furthermore, the asynchrony assumption makes it very difficult to design and verify distributed algorithms, whereas the perfect synchrony assumption is very expensive, if not impossible, to implement in real distributed systems.

Based on these observations, researchers (e.g., [1, 2, 4, 5, 8, 9, 10, 11, 17]) began to investigate the impact on distributed computing if those timing assumptions are relaxed or tightened to some extent in order to reflect more realistic situations. The new timing models that are obtained by relaxing or tightening the two extreme timing assumptions are called *partially synchronous models*.

*Received by the editors August 30, 1999; accepted for publication (in revised form) December 16, 2002; published electronically June 25, 2003. A preliminary version of this paper appeared as [18]. Much of this work was done while the authors were with the Department of Computer Science, University of North Carolina at Chapel Hill. This work was supported by an IBM Faculty Development Award, NSF Presidential Young Investigator Award CCR-9158478, and TAMU Engineering Excellence funds.

http://www.siam.org/journals/sicomp/32-4/36094.html

†Department of Computer Science, North Carolina State University, Raleigh, NC 27695-7534 (rhee@csc.ncsu.edu).

‡Department of Computer Science, Texas A&M University, College Station, TX 77843-3112 (welch@cs.tamu.edu).

The goal of this paper is to compare the computational power of a family of partially synchronous models by studying the time needed to solve a distributed computing problem called the *session problem*.

1.1. The session problem. The (s, n)-session problem was first presented in [3] and further studied in [5]. Informally, a *session* is a minimal-length computation fragment that involves at least one special "synchronization" step by every process in a distinguished set of n processes. An algorithm that solves the (s, n)-session problem must guarantee that in every computation there are at least s disjoint sessions and that eventually all the n processes become idle.

The (s, n)-session problem is an abstraction of the synchronization used in many distributed computing settings. The (s, n)-session problem, like the mutual exclusion and dining philosophers problems, concerns possible ordering of process events (e.g., a process finishing its assigned task) rather than the computation of particular outputs.

Consider, for example, *barrier synchronization* [13], a fundamental mechanism in concurrent systems which guarantees that all processes have finished a specified task in their execution before any proceeds. Barrier synchronization is a special case of the (s, n)-session problem when $s = 2$, and solutions for the $(2, n)$-session problem can be used to construct barrier synchronization: after each process finishes its specified task, it keeps taking synchronization steps until the $(2, n)$-session algorithm terminates.

As discussed in [5], another example of the (s, n)-session problem can be found in a distributed linear equation solver, where each process holds part of the input data (cf. [7]) and iterates to solve equations by relaxation. Each process takes one synchronization step when it changes its data. Sufficient interleavings of synchronization steps by different processes ensure a correct output since they imply sufficient interaction among the intermediate values computed by the processes.

The (s, n)-session problem is also an abstraction of a simple message distribution system in which a sending process writes a sequence of s messages one at a time on a board (e.g., port or mailbox) visible to all and waits after each message until all $n-1$ other processes have read the message before writing the next one. Each reading step by a process is one synchronization step of the process. Any protocol which ensures that the sender has waited sufficiently long solves the (s, n)-session problem.

Since the time complexity of the session problem is very sensitive to the timing assumptions of the underlying model, it has been used as a test case to demonstrate the theoretical differences in the time needed to solve problems in various timing models [3, 5, 16, 18, 19]. Using the session problem, we can quantify differences between various models in terms of the time complexity needed to solve distributed computing problems. Precise time complexities for various timing models allow us to show complexity gaps among the models. Time complexity gaps can provide valuable information to system designers in evaluating and comparing the various timing models and deciding what timing guarantees they have to provide or do not have to provide to build efficient yet cost-effective distributed systems.

A solution for the (s, n)-session problem normally involves several methods for counting sessions during execution. In particular, the first or last session is often counted in a different way than the other sessions. Hence, the time complexity of a solution is usually expressed as a function of s, n and some additional terms to account for the complexity of counting the first or last session. When we qualitatively evaluate relative time complexities of different timing models, the term associated with s has more weight than the other terms in deciding the time complexity hierarchy.

1.2. Timing models. We consider two different interprocess communication models: *shared memory* (SM) and *message passing* (MP). In the SM model, processes communicate only by means of shared variables.

In the MP model, communication is done by exchanging messages across a network. A process can broadcast a message at a step; the message is guaranteed to be delivered to every process after some finite time.

Process step time is the amount of time between two consecutive steps of the same process, and *message delay* is the amount of time between when a message is sent and when the message is received. The relevant timing parameters of a model are the minimum step time, c_1, the maximum step time, c_2, and additionally, for the MP model, the minimum message delay, d_1, and the maximum message delay, d_2.

We consider families of timing models for both SM and MP systems. The timing models are obtained by considering independently whether each parameter is known (i.e., can be hard-wired into the processes' code) or unknown, giving rise to four SM models and 16 MP models. Some of these models have been studied previously in both practical and theoretical contexts.

Models with known maximum and minimum step times are commonly called *semisynchronous models* and have been previously studied for various distributed computing problems, including the consensus problem, the mutual exclusion problem, and the session problem (see [1, 2, 4, 5, 6, 8, 14, 15, 17, 18, 19]). The semisynchrony models systems where information about timing parameters, such as process step time, is only approximately known; e.g., processes may have access to inaccurate clocks that operate at approximately, but not exactly, the same rate.

Models with unknown step times have been studied for the consensus problem [11] and for the mutual exclusion problem [1]. As those papers argue, these models provide a useful abstraction of the timing constraints in real systems.

Models in which the minimum step time is known but the maximum step time is unknown abstract event-driven processing such as responding to user inputs or nonperiodic device interrupts [18]. In these models, processes can be blocked for an arbitrarily long (but finite) time waiting for a certain condition to be true or a certain event to occur but cannot take two consecutive steps faster than a certain amount of time.

A lower bound result or impossibility result shown for an asynchronous model does not automatically carry over to a model with unknown bounds. For instance, the work on the consensus problem in [11] showed that fault-tolerant consensus can be solved in a model with unknown bounds, although it cannot be solved in an asynchronous system [12]. There is more leeway in constructing "bad" executions in an asynchronous system than there is in one with unknown bounds. Thus, it is worth investigating how knowledge of step time and message delay affects the session problem.

1.3. Previous work on the session problem. The upper and lower bounds on the time required to solve the session problem in an asynchronous SM system shown by Arjomandi, Fischer, and Lynch [3] demonstrated the first such case where asynchronous systems are less efficient than synchronous systems. In the synchronous model, all processes run in lockstep, while in the asynchronous model no bounds on process running rates exist. Their result showed an inherent time complexity gap between the synchronous and asynchronous models: s steps are sufficient for s sessions in the synchronous model, i.e., no interprocess communication is needed, but $(s-1)\lfloor \log_a n \rfloor$ steps are necessary for the asynchronous model. The $\lfloor \log_a n \rfloor$

factor is essentially the cost of communication, where a is the maximum number of distinct processes that are ever allowed to access any given shared variable. Thus, one interprocess communication per session is needed in the asynchronous SM model.

Attiya and Mavronicolas [5] show a similar result for an asynchronous MP system in which there is a maximum message delay d_2 but the minimum message delay d_1 is zero. Their results show that the asynchronous MP model requires $(s-1)\cdot d_2$ time to solve the (s,n)-session problem (at least one message delay per session).

The session problem has been studied in a semisynchronous model as well, in which there are known minimum and maximum step times and there is a (not necessarily known) maximum message delay. The upper bound in the MP model shown by Attiya and Mavronicolas [5] is $(s-1)\cdot\min\{\frac{c_2}{2c_1}c_2, d_2\}$. A nearly matching lower bound (within a factor of 2 of the upper bound) also appears in [5]. These results imply that the efficiency of the semisynchronous SM model lies between those of the synchronous and asynchronous models.

In a *periodic* model where processes run at a fixed unknown periodic rate, nearly matching lower and upper bounds shown by [18, 19] indicate that at least one communication is required to solve the session problem. These bounds also indicate the inherent cost of synchronizing periodically running processes and the existence of time complexity gaps among the synchronous, periodic, and asynchronous timing models.

1.4. Our results. Our complexity results are organized around "ways to count" s sessions in a computation. The intuition is that processes must have some way to count the passage of the other processes' steps in order to "know" when a session has occurred.

Note that $s\cdot c_2$ is an obvious lower bound for all models because each process has to take at least s steps to solve the (s,n)-session problem and each step takes up to c_2 time [3]. We omit from the discussion the obvious lower bound $s\cdot c_2$.

1.4.1. Shared memory results. In order for our results to be comparable to prior work, we study SM systems with a constant parameter a, which is the maximum number of distinct processes that are ever allowed to access any given shared variable. When a is smaller than the total number of processes in the system, it is not possible for all processes to exchange information in a single step. Instead, information must be propagated from process to process. Thus, as a gets smaller, the amount of propagation required increases. The motivation for this restriction on communication comes from the fact that in a distributed SM system, some part of memory is local to a process and can be accessed quickly, while the rest is remote and requires more time for accesses.

Table 1.1 summarizes our results on the time complexity of solving the (s,n)-session problem in SM models.

Our results indicate that if either the minimum or maximum step time (or both) is unknown, then the running time for the (s,n)-session problem is $(s-1)\cdot c_2\cdot\Theta(\log n)$, i.e., roughly one communication cost $(c_2\cdot\Theta(\log n))$ is required for each session. On the other hand, if both step times are known, then the running time is $(s-1)\cdot c_2\cdot\min\{\frac{c_2}{2c_1}, \Theta(\log n)\}$. In this model, processes can use timing information about relative step times to count locally in order to determine when enough sessions have elapsed. We call this counting technique the *step time* (ST) *method*. It was first proposed in [5] for the semisynchronous MP model. However, if the gap between the minimum and maximum step times is sufficiently large, then it is more cost-effective to use explicit communication. We call this counting technique the *explicit communication* (EC) *method*. It was first proposed in [3] for the asynchronous SM model.

TABLE 1.1
Time bounds for the (s,n)-session problem in SM models; a is the maximum number of processes that can access a shared variable, c_1 and c_2 are the minimum and maximum step times.

c_1	c_2	Lower bound	Upper bound
unknown	unknown	$(s-1)\cdot c_2\cdot \log_a n$	$(s-1)\cdot c_2\cdot \Theta(\log n)$
unknown	known	$(s-1)\cdot c_2\cdot \log_a n$	$(s-1)\cdot c_2\cdot \Theta(\log n)$
known	unknown	$(s-1)\cdot c_2\cdot \log_a n$	$(s-1)\cdot c_2\cdot \Theta(\log n)$
known	known	$(s-1)\cdot c_2\cdot \min\{\frac{c_2}{2c_1}, \log_a n\}$	$(s-1)\cdot c_2\cdot \min\{\frac{c_2}{2c_1}, \Theta(\log n)\}$

These results are analogous to those of [3]: intuitively, if either bound is unknown, then the system can be considered somewhat "asynchronous"; otherwise the system behaves "more synchronously." As we discussed in the introduction, the asynchronous lower bound of [3] does not automatically imply any of the lower bounds in the unknown bound models; however, it is the case that the proof in [3] also works in the case where both bounds are unknown. Mavronicolas [16] independently and concurrently also developed the same bounds for the model where both minimum and maximum step times are known.

1.4.2. Message passing results. In the MP case, we discovered a pattern of upper bounds consisting of eight different groups of models. As in the SM case, the pattern is based on different counting methods. However, there are three additional counting methods available in message passing, so the relationships are more involved.

In the following, we specify each model by a tuple (c_1, c_2, d_1, d_2). Each entry in a tuple is a real value if that parameter is known, and "?" if it is unknown. For example, we denote the model in which only the maximum step time is known by $(?, c_2, ?, ?)$.

In addition to the two counting methods available in the SM model (EC and ST), three other counting methods are used in the MP model: (1) The *message delay* (MD) *method* uses the known difference between the minimum and maximum message delays; (2) *combination method* 1 (CB1) uses the known minimum step time in combination with the difference between the minimum and maximum message delays; and (3) *combination method* 2 (CB2) uses the known maximum message delay in combination with the known minimum step time.

Table 1.2 shows the approximate per-session cost for each counting method that is applicable when specific timing information about the system is available. The upper bound on the time complexity for a particular timing model is the minimum, over all applicable counting methods, of the time complexity of the counting methods.

These counting methods divide the models into eight groups, as shown in Table 1.3. Figure 1.1 shows a lattice of timing models based on the counting methods that a model can use. In the figure, an arrow from one group to another means that the time required to compute a session in the models comprising the source group is asymptotically equal to or larger than the time required to compute a session in the models comprising the target group. For instance, the arrow from G1 to G2 indicates that computing a session in G1 takes at least as much time as doing so in G2.

The results for group G4, when step time bounds are known, were previously shown by Attiya and Mavronicolas [5]. All the remaining results are new; as mentioned before, the asynchronous results in [5] do not automatically imply the same results in the unknown bound cases, although the proof techniques are similar.

We show that the upper bounds on the time complexity for the timing models are asymptotically optimal. Some of our lower bounds require certain relationships to

TABLE 1.2

The approximate per-session cost of each counting method used to solve the session problem when the required timing knowledge is available; c_1 and c_2 are the minimum and maximum step times, d_1 and d_2 are the minimum and maximum message delays, and $u = d_2 - d_1$ is the uncertainty in message delay.

Counting methods	Required knowledge	Approximate per-session cost
Explicit communication (EC)	none	d_2
Step times (ST)	c_1, c_2	$\frac{c_2}{c_1} c_2$
Message delays (MD)	d_1, d_2	$\frac{d_2}{d_1} u$
Combination 1 (CB1)	c_1, d_1, d_2	$\frac{c_2}{c_1} u$
Combination 2 (CB2)	c_2, d_1	$\frac{d_2}{d_1} c_2$

TABLE 1.3

Groups of models that can use the same counting methods.

Group	Models	Usable counting method(s)
$G1$	$(?,?,?,?), (?,?,?,d_2), (?,?,d_1,?), (?,c_2,?,?),$ $(?,c_2,?,d_2), (c_1,?,?,?), (c_1,?,?,d_2), (c_1,?,d_1,?)$	EC
$G2$	$(?,?,d_1,d_2)$	EC, MD
$G3$	$(?,c_2,d_1,?)$	EC, CB2
$G4$	$(c_1,c_2,?,?), (c_1,c_2,?,d_2)$	EC, ST
$G5$	$(c_1,?,d_1,d_2)$	EC, MD, CB1
$G6$	$(?,c_2,d_1,d_2)$	EC, CB2, MD
$G7$	$(c_1,c_2,d_1,?)$	EC, CB2, ST
$G8$	(c_1,c_2,d_1,d_2)	EC, CB2, CB1, ST, MD

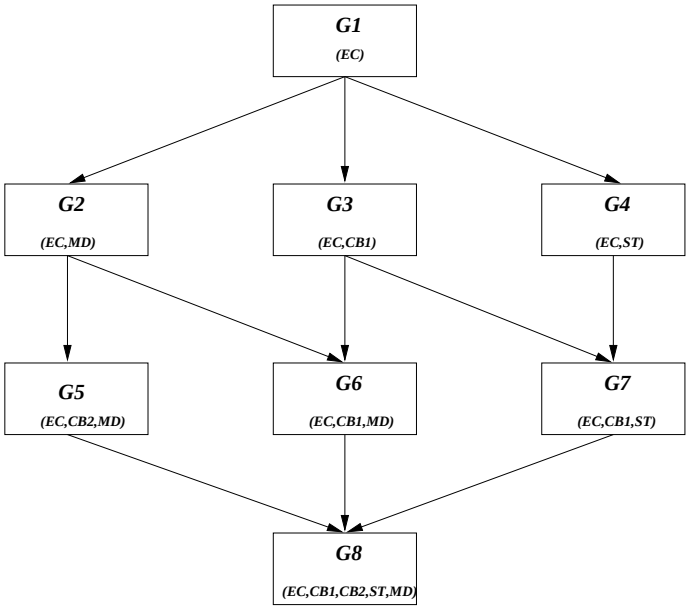


FIG. 1.1. *A lattice of model groups can be formed based on the counting methods (shown in parenthesis) and tight time complexity bounds of the timing models.*

hold between some of the parameters. For example, consider the model in which only c_2 and d_1 are known. The only applicable counting methods are EC and CB2. Thus, the per-session cost is (approximately) $\min\{d_2, \frac{d_2}{d_1} \cdot c_2\}$. The lower bound we prove for this model gives a per-session cost of approximately $\frac{2d_2}{3d_1} \cdot c_2$, assuming $2c_2 \leq d_1$. Since $2c_2 \leq d_1$, algebraic manipulation shows that this lower bound is less than d_2. Thus our upper and lower bounds are asymptotically tight if $2c_2 \leq d_1$.

As in the case of the SM models, the general trend of these bounds is that if a smaller number of the parameters in a model are known, the model behaves more like "asynchronous," and otherwise, more like "synchronous."

As the time complexity gaps (i.e., the difference between the upper bound in a model and the lower bound in another model) among the models sometimes overlap, it is rather difficult to analyze the relative strength qualitatively without making assumptions on parameters. However, when specific values for each known parameters are given, the actual bounds can be used to analyze the relative strength of the models quantitatively.

As process step times become more synchronous (i.e., $c_1 \simeq c_2$) and message delays become erratic (i.e., $c_2 \ll d_1 \ll d_2$), the general trend of the bounds is that $\{G1, G2, G5\} > \{G3, G4\} > \{G6, G7, G8\} > S$, where S is the synchronous model and ">" denotes that it takes more time to solve the session problem. As message delays become more synchronous and smaller (i.e., $d_1 \simeq d_2$ and $d_2 \geq c_2^2/2c_1$) and process step times become more erratic (i.e., $c_1 \ll c_2$), the trend is that $\{G1, G6\} > \{G2, G5, G3, G4, G7, G8\} > S$. These trends suggest that when process step times are fairly "synchronous," the ST method can be more cost-effective than MD, CB2, and CB1, while when process step times are more "asynchronous" than message delays, the opposite is true.

1.4.3. Proof techniques. We unify the lower bound proof techniques of [3] in the SM model and [5] in the MP model into one "modular" lower bound proof. Our technique is unique in that, instead of obtaining a lower bound for each model independently, we develop one sufficient condition for any given lower bound to hold in any given timing model. This sufficient condition consists of a set of algebraic relations involving (1) the timing parameters of the given model; (2) the given lower bound; and (3) some input parameters that need to be provided to prove the lower bound. Testing whether a lower bound holds in a timing model is a simple algebraic exercise of finding those input parameters that satisfy the relations.

The upper bounds are also obtained in a modular way. We first find algorithms (i.e., ways to count sessions) that work correctly when a certain set of timing parameters is known. Since several algorithms can be applicable to a model, we provide a scheme to combine these algorithms without increasing the time complexity of any of its applicable algorithms. The resulting upper bound of a model is simply the minimum of the time complexity of all the algorithms applicable to the model.

1.5. Organization. The rest of this paper is organized as follows. Section 2 gives the definition of the system model. Section 3 contains our modular lower bound result for the time complexity of the session problem. Our algorithmic counting methods are presented in section 4. Section 5 draws together the results for shared memory and section 6 does the same for message passing. We conclude in section 7.

2. Definitions.

2.1. Systems. The system model definition is similar to that defined in [3].

There are finite sets P of processes and V of shared variables. A *process* has a set of internal states, including an initial state. Each *shared variable* has a set of values that it can contain, including an initial value. A *global state* is a tuple of internal states of each process, and values of each shared variable. The *initial* global state contains the initial state for each process and the initial value for each shared variable.

A process can both read and write a shared variable in a single atomic step (i.e., the variable supports read-modify-write operations); we do not assume any upper bound on the size of the variables. A *step* π consists of simultaneous changes to the state of some process p and the value of some set of variables $x_1, \ldots, x_k$ (for some integer k), where p is allowed to access x_i, $1 \leq i \leq k$, depending on the current state of that process and current values of the variables. More formally, we represent the step π with a tuple $((q, p, r), (u_1, x_1, v_1), \ldots, (u_k, x_k, v_k))$, where q and r are old and new states of a process $p \in P$, and u_i and v_i are old and new values of a shared variable $x_i \in V$. We define $proc(\pi) = p$ and $var(\pi) = \{x_1, \ldots, x_k\}$. We say that step π is *applicable* to a global state if p is in state q and x_i has value u_i for all i in the global state.

An *algorithm* consists of P, V, and set Σ of possible steps. For all processes $p \in P$ and all global states g, there must exist some step in Σ involving process p that is applicable to global state g. This condition ensures that p never blocks. A *computation* of a system is a sequence of steps $\pi_1, \pi_2, \ldots$ such that (1) π_1 is applicable to the initial global state; (2) each subsequent step is applicable to the global state resulting from the previous step; and (3) if the sequence is infinite, then every process takes an infinite number of steps. That is, there is no process failure.

A *timed computation* (α, T) of a system is a computation $\alpha = \pi_1, \pi_2, \ldots$ together with a mapping T from positive integers to nonnegative real numbers that associates a real time with each step in the computation. T must be nondecreasing and, if the computation is infinite, increase without bound. This way of modeling processes assumes that the time taken for local computation at a step is negligible.

2.1.1. SM model. We specialize the general system into the SM system in which processes communicate with each other by means of shared variables. Each step π involves only one shared variable. Associated with each variable is a set of at most a processes that are allowed to access that variable.

2.1.2. MP model. We specialize the general system into the MP system, in which processes communicate with each other by exchanging messages. P consists of the *regular* processes, denoted by the set R, plus a distinguished process N, called the *network*. The network schedules the delivery of messages sent among the regular processes. V, the set of shared variables, equals $\{net\} \cup \{buf_p : p \in R\}$, where the values taken on by each variable are sets of messages. The variable *net* models the state of the network, i.e., the set of messages in transit. The variable buf_p holds the set of messages that have been delivered to p by the network but not yet received by p.

A step of a process p in R consists of p receiving the set M of messages in its buffer buf_p and based solely on those messages and its current state, changing its local state and sending out some message m to all the regular processes. More formally, the result of the step is to set buf_p to empty (i.e., receive messages), to add (m, q) to

net for all q in R (i.e., send a message), and to change state. So, the step involves two shared variables, buf_p and *net*. A step of N is to deliver some message of the form (m, q) in *net* to q. More formally, the result of the step is to remove (m, q) from *net* and add m to buf_q. We call this step the *delivery step* of m. Accordingly, the step also involves two shared variables, *net* and buf_q. We define $msg(\sigma)$ to be the message that is involved in a step σ of a process in P.

This definition of the MP model is an abstract model of a reliable strongly connected network with any topology (i.e., for every pair of processes, there exists a communication path between the two processes).

In a timed computation, each message has a *delay*, defined to be the difference between the time of the step that adds it to *net* and the time of the step that removes it from *net*. If the message is never removed, then it has infinite delay. The delay only counts the time in transit in the network and does not include the time that the recipient takes to receive the message. Note that after a message is delivered to a destination process p, p has to take at least one step to receive the message. That is, the time elapsed between the delivery step of a message m and the step of the destination process which finally removes m from the buffer is not counted toward the message delay.

2.2. Timing models. First we consider SM models. Let v and w be two positive real numbers with $v \leq w$. $M(v, w)$, called a *submodel*, is the set of all timed computations of a system in which all step times (the time between two consecutive steps by the same process) are within $[v, w]$.

An SM model is specified by indicating whether the minimum and maximum step times are known, and if so, what their values are. Formally, an *SM model* is denoted $\mathcal{M}[c_1, c_2]$, where $c_i \in \{?\} \cup \mathcal{R}^+$ ($\mathcal{R}^+$ is the set of positive reals). If the minimum step time is known, then c_1 is some positive real; otherwise, $c_1 = ?$. If the maximum step time is known, then c_2 is some positive real; otherwise, $c_2 = ?$.

We define the four SM models of interest as follows:

$$\begin{aligned}
\mathcal{M}[c_1, c_2] &= \{M(c_1, c_2)\} \quad \text{if } c_1 \in \mathcal{R}^+ \text{ and } c_2 \in \mathcal{R}^+.\\
\mathcal{M}[c_1, ?] &= \{M(c_1, w) : w \geq c_1\} \quad \text{if } c_1 \in \mathcal{R}^+.\\
\mathcal{M}[?, c_2] &= \{M(v, c_2) : 0 < v \leq c_2\} \quad \text{if } c_2 \in \mathcal{R}^+.\\
\mathcal{M}[?, ?] &= \{M(v, w) : 0 < v \leq w\}.
\end{aligned}$$

We now consider MP models. Let v, w, x, and y be four positive real numbers with $v \leq w$ and $x \leq y$. $M(v, w, x, y)$, called a *submodel*, is the set of all timed computations in which all step times are within $[v, w]$ and all message delays are within $[x, y]$.

The 16 MP models are defined analogously to the four SM models. For example,

$$\mathcal{M}[?, c_2, d_1, ?] = \{M(v, c_2, d_1, y) : 0 < v \leq c_2 \text{ and } y \geq d_1\} \quad \text{if } c_2 \text{ and } d_1 \in \mathcal{R}^+.$$

We number the models 0 through 15 using the binary representation, assuming a parameter that equals ? is replaced with 0 and otherwise with 1. For instance, $\mathcal{M}[?, c_2, d_1, ?]$ is numbered $0110_2 = 6$.

We say that a timed computation α is *admissible* for a submodel M if α is in M, and is *admissible* for a model $\mathcal{M}$ if α is admissible for some M in $\mathcal{M}$.

2.3. The (s, n)-session problem. We now state the conditions that must be satisfied for a system to *solve the* (s, n)*-session problem*.

There is a distinguished set Y of n shared variables called *ports*; Y is a subset of V in SM models; and Y is the set of *buf* variables in MP models. There is a unique process in P (in R in MP models) corresponding to each port, which is called a *port process*, and no two port processes can be assigned to the same port. A *port step* is any step involving a port and its corresponding port process. A port can be accessed by processes in addition to its corresponding port process, but such a step is not a port step. There may be some processes which are not port processes; i.e., it is possible for $|P|$ to be larger than n.[1]

Each port process in P must have a subset of special states, called *idle* states. The set Σ of steps of the system must guarantee that once a process is in an idle state, it always remains in an idle state, and after a process enters an idle state, it does not access a port.

A *session* is a minimal sequence of steps containing at least one port step for each port in Y. A computation *performs* s sessions if it can be partitioned into s segments, each of which is one session. Every infinite admissible timed computation must perform at least s sessions and eventually all port processes must be in idle states.

2.4. Time complexity. We give the definitions for the SM models. The time complexity definitions for MP models are analogous to those for SM models.

An algorithm A in a submodel $M(x, y)$ has *running time* t if t is the maximal time, over all admissible computations of A for $M(x, y)$, until all port processes become idle.

Let f be a function from $\mathcal{R}^+ \times \mathcal{R}^+$ to $\mathcal{R}^+$. We abuse notation and say that an algorithm A in model $\mathcal{M}[c_1, c_2]$ has *upper bound* $f(c_1, c_2)$ if A has running time at most $f(x, y)$ in every submodel $M(x, y)$ in $\mathcal{M}[c_1, c_2]$. (This is an abuse of notation because c_1 or c_2 might equal ? instead of being a positive real constant.)

$\mathcal{M}[c_1, c_2]$ has *lower bound* $f(c_1, c_2)$ if for every algorithm A, there is a submodel $M(x, y)$ such that A has running time at least $f(x, y)$ in that submodel.

3. Modular lower bound. In this section, we give a modular lower bound proof that holds for all the timing models, both SM and MP. Our lower bound proof is motivated by the proofs in [3] and [5]. Our technique is unique in that, instead of obtaining a lower bound for each model independently, we develop a sufficient condition for a lower bound to hold in any given timing model. This sufficient condition consists of a set of algebraic relations on (1) the timing parameters of the given model; (2) the given lower bound; and (3) some other input parameters (shown below). Thus, testing whether a lower bound holds in a timing model is a simple algebraic exercise of finding those input parameters that satisfy the relations. The theorem below proves the sufficient condition; in its statement, c, c'_1, c'_2, d'_1, d'_2, and B are the input parameters, and SC1 to SC3 and MC1 to MC5 are the algebraic relations. In the proof of the theorem, we present some intuitive ideas behind the theorem and then formalize the ideas.

THEOREM 3.1. *Let M be a timing model that satisfies the following.*

If $M = \mathcal{M}[c_1, c_2]$ is an SM model, then there exist positive real numbers c, c'_1, c'_2 and a function f with $B = f(c'_1, c'_2)$ such that

[1] This possibility is implicitly contained in [3], which refers to making the port processes the leaves of a tree network.

$$\begin{array}{lllll}
\text{SC1.} & (B \le c_2' \cdot \log_a n) & \wedge & (c_1' \le c_2'). & \\
\text{SC2.} & (c_1' \le \frac{1}{2}c) & \wedge & ((c_1' = c_1) & \text{if } c_1 \ne ?). \\
\text{SC3.} & (c_2' \ge B\frac{c}{c_2'}) & \wedge & ((c_2' = c_2) & \text{if } c_2 \ne ?).
\end{array}$$

If $M = \mathcal{M}[c_1, c_2, d_1, d_2]$ is an MP model, then there exist positive real numbers c, c_1', c_2', d_1', d_2' and a function f with $B = f(c_1', c_2', d_1', d_2')$ such that

$$\begin{array}{lllll}
\text{MC1.} & (B \le d_2') & \wedge & (c_1' \le c_2') \wedge (d_1' \le d_2'). & \\
\text{MC2.} & (c_1' \le \frac{1}{2}c) & \wedge & ((c_1' = c_1) & \text{if } c_1 \ne ?). \\
\text{MC3.} & (c_2' \ge B\frac{c}{c_2'}) & \wedge & ((c_2' = c_2) & \text{if } c_2 \ne ?). \\
\text{MC4.} & (d_1' \le (d_2' - B)\frac{c}{c_2'} + c) & \wedge & ((d_1' = d_1) & \text{if } d_1 \ne ?). \\
\text{MC5.} & (d_2' \ge (d_2' + B)\frac{c}{c_2'} - c) & \wedge & ((d_2' = d_2) & \text{if } d_2 \ne ?).
\end{array}$$

Then a lower bound on the time complexity of the (s, n)-session problem for M is $(s-1) \cdot f(c_1, c_2)$ if M is an SM model, and $(s-1) \cdot f(c_1, c_2, d_1, d_2)$ if M is an MP model.

3.1. Proof of Theorem 3.1.

Informal description. By way of contradiction we assume that there exists such an algorithm A that solves the (s, n)-session problem in model M within less time than the stated lower bound. We prove that there exists an infinite timed computation of A that is admissible for M yet contains fewer than s sessions, contradicting the assumed correctness of A.

More specifically, we first fix a submodel M' of M and pick an infinite timed computation (α, T) of A that is admissible for M'. Then we retime and reorder some steps in α to obtain a new infinite timed computation (α', T') that has only $s-1$ sessions, yet is admissible for some submodel M'' of M. (M' and M'' are not necessarily the same.)

Real numbers c_1', c_2', d_1', and d_2' are the minimum and maximum step times and message delays of submodel M', respectively, for which (α, T) is admissible. Real numbers $\frac{1}{2}c$, $B\frac{c}{c_2'}$, $(d_2' - B)\frac{c}{c_2'} + c$, and $(d_2' + B)\frac{c}{c_2'} - c$ are the minimum and maximum step times and message delays of M'' for which (α', T') is admissible.

The conditions in the theorem statement are used to prove that M' and M'' really are submodels of M. Below we provide some intuition for these conditions.

1. B is roughly the time for a process to "recognize" one session in a computation. The first clause in conditions SC1 and MC1 states that B does not take more than the lower bound on the maximum communication delay in the SM and MP models.
2. Conditions SC1 and MC1 ensure that minimum and maximum step times and message delays (c_1', c_2', d_1', and d_2') in submodel M' satisfy the property that the minimum is not larger than the maximum.
3. Conditions SC2 and MC2 ensure that, if the minimum step time (c_1) is known in M, then in M' it is equal to that of M, and in M'' it is at least as large as that of M.
4. Conditions SC3 and MC3 ensure that, if the maximum step time (c_2) is known in M, then in M' it is equal to that of M, and in M'' it is not larger than that of M.
5. Condition MC4 ensures that, if the minimum message delay (d_1) is known in M, then in M' it is equal to that of M, and in M'' it is at least as large as that of M.

6. Condition MC5 ensures that, if the maximum step time (d_2) is known in M, then in M' it is equal to that of M, and in M'' it is not larger than that of M.

Since B is roughly the upper bound on the time to have one session, the time complexity lower bound to solve the (s,n)-session problem cannot be less than $(s-1)$ times the maximum B, which is determined by the timing parameters of M.

We now need to prove that satisfying the above conditions is sufficient to prove the time complexity lower bound for solving the (s,n)-session problem. This involves several procedures.

1. We prove that (α, T) is admissible for M', and M' is a submodel of M. In (α, T), processes enter an idle state before $B \cdot (s-1)$.
2. We then reorder steps in α to obtain α' without violating the causal dependency among process steps. The causal dependency among two process steps happens, for example, because one step receives a message sent by the other step. Thus, for example, α' should not order the receive step before the send step. This procedure involves several other steps.
   (a) We first break α into two segments. The first segment, β, is up to the last step taken by any process before all processes enter the idle state. The second segment, γ, is the rest of α. It is clear that β should contain s sessions because α is a computation of A that solves the (s,n)-session problem.
   (b) We then break β into $s-1$ equal nonoverlapping segments of time period B. We reorder the process steps only within each segment without violating their causal dependency so that each segment by itself does not contain one session. Since each segment contains less than one session, the reordered sequence β' does not contain more than $s-1$ sessions. Since each segment does not violate the causal dependency, all the port processes will be in the same state as they are in the corresponding segment in β. Thus, in the end of β', all the port processes are in the same state as in β, and $\alpha' = \beta'\gamma$ is an admissible computation for M because $\alpha = \beta\gamma$ is.
3. Reordering steps perturbs the timings of process steps. We show a timing mapping T' for β' whose minimum and maximum step times are $c/2$ and $B\frac{c}{c_2'}$, and whose minimum and maximum message delays are $(d_1' \leq (d_2' - B)\frac{c}{c_2'} + c)$ and $(d_2' \geq (d_2' + B)\frac{c}{c_2'} - c)$. The conditions in the theorem are used to show that these retimed parameters are within the constraints of a submodel M'' of M.
4. Since there is a timed computation (α', T') that is admissible for a submodel of M which contains less than $s-1$ sessions, this is a contradiction.

Formal description. We now formalize these ideas.

Let M' be the submodel of M that has minimum and maximum step times and message delays equal to c_1', c_2', d_1', and d_2', respectively, where the following hold: (a) $c_1' \leq c_2'$; (b) $c_1' = c_1$ if the minimum step time of M (c_1) is known; and (c) $c_2' = c_2$ if the maximum step time of M (c_2) is known. Furthermore, if M is an MP model, (d) $d_1' \leq d_2'$; (e) $d_1' = d_1$ if the minimum message delay of M (d_1) is known; and (f) $d_2' = d_2$ if the maximum message delay of M (d_2) is known.

Since M' is a submodel of M, by the assumption that algorithm A is correct for M, A has running time in M' less than $(s-1) \cdot B$, where $B = f(c_1', c_2')$ if M is an

SM model, and $B = f(c_1', c_2', d_1', d_2')$ if M is an MP model.

Let (α, T) be the infinite timed computation of A in which all the regular processes take steps at the same speed in round robin order and each process's ith step occurs at time $i \cdot c_2'$. Furthermore, if M is an MP model, all the message delays in (α, T) are exactly d_2'. Note that (α, T) is admissible for M' (and thus for M).

Let $\alpha = \beta\gamma$, where β contains all the steps that occur in (α, T) during time interval $[0, (s-1) \cdot B)$ and γ contains the rest of α. Note that all the states in γ are idle states because A solves the (s, n)-session problem in time less than $(s-1) \cdot B$.

Case 1. If $B \leq c_2'$, then β contains only $s-1$ sessions because each process takes only $s-1$ steps in β. This is a contradiction since all regular processes are in an idle state in γ.

Case 2. For the rest of the proof, assume that $B > c_2'$. For convenience of presentation, we assume that B is divisible by c_2'.[2]

We will reorder and retime (i.e., assign new times to) steps in β to obtain (β', T'). To ensure that the retimed computation leads to the same global state, this retiming should not violate the dependencies among process steps. Informally, a dependency arises between two steps if they are steps of the same process; if one step reads a variable previously accessed by the other; or if one step is the receipt of a message sent by the other. A more precise definition of dependency is given below.

We construct a partial order $\leq_\beta$ on the steps in β, representing dependency. Let $\sigma \leq_\beta \tau$ for every pair of steps σ and τ in β, and say that τ *depends* on σ if

- $\sigma = \tau$, or
- $T(\sigma) < T(\tau)$ and $proc(\sigma) = proc(\tau) \neq N$, or
- $T(\sigma) < T(\tau)$ and $msg(\sigma) = msg(\tau)$ if M is an MP model, or
- $T(\sigma) < T(\tau)$ and $var(\sigma) = var(\tau)$ if M is an SM model.

Close $\leq_\beta$ under transitivity.

Let $\beta = \beta_1 \ldots \beta_{s-1}$ such that each β_k, which we call *segment* k, $1 \leq k \leq s-1$, consists of all the steps in (α, T) during time interval $[(k-1)B, kB)$. Note that in an MP model, no message sent in β_k is received in β_k since $B \leq d_2'$ by MC1 and d_2' is the message delay in (α, T).

Informally speaking, we will reorder steps in each segment β_k without violating the dependencies as follows. We first pick one port variable for each segment in such a way that the same port variable is not picked for two consecutive segments. Let y_i be the port variable picked for segment β_i. Then we reorder the steps of each segment, resulting in two "subsegments" such that the first subsegment does not contain any port event accessing y_i and the second subsegment does not contain any port event accessing y_{i+1}. The reordered sequence will contain only $s-1$ sessions. Figure 3.1 illustrates an example when $s = 4$.

However, for the reordered computation to end in the same state as β, this reordering should not violate relation $\leq_\beta$ within each segment (relation $\leq_\beta$ is not violated across segments by the reordering because steps are not reordered out of their own segments). Thus, we must choose the port y_k for each segment β_k such that the first step in β_k to access y_{k-1} does not depend on the last step in β_k to access y_k. The following claim shows that this can be done.

CLAIM 3.2. *Let y_0 be an arbitrary port in Y. For all k, $1 \leq k \leq s-1$, there exists a port variable y_k such that it is false that $\tau_k \leq_\beta \sigma_k$, where τ_k is the first step in β_k that accesses y_{k-1} and σ_k is the last step in β_k that accesses y_k.*

[2] If B is not divisible by c_2', the lower bound we obtain is $(s-1) \cdot \lfloor \frac{f(c_1, c_2)}{c_2} \rfloor \cdot c_2$ instead of $(s-1) \cdot f(c_1, c_2)$ in the SM case, and similarly for the MP case.

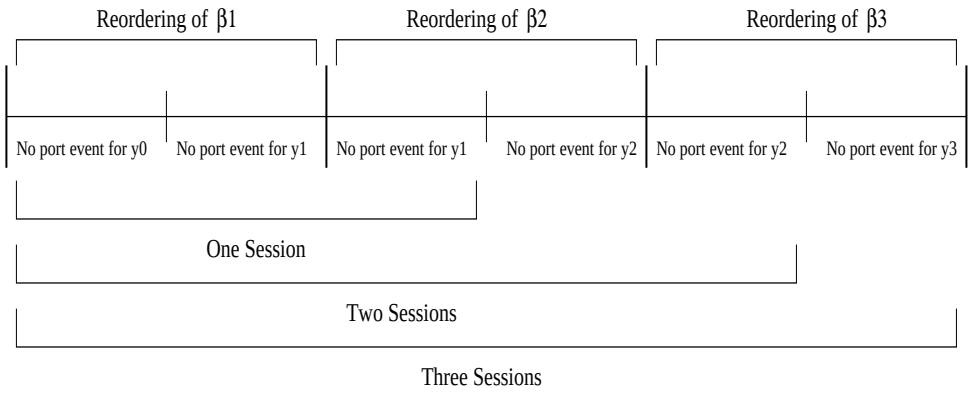


FIG. 3.1. *An example of reordering when* $s = 4$.

Proof. If M is an MP model, no message sent in β_k is delivered in β_k because the size of each segment is less than d_2', and d_2' is the message delay. Because there is no $\leq_\beta$ relation between any two steps of different processes in β_k, any port variable except y_{k-1} can be chosen as y_k.

If M is an SM model, part of the proof of Theorem 2 of [3][3] proves that there exists such y_k if each process takes fewer than $\log_a n$ steps in a segment. Because in β_k each process takes fewer than $\log_a n$ steps (cf. SC1), there exists such y_k in β_k. □

Claim 3.2 allows us to use the y_k's to reorder β to obtain a new sequence with less than s sessions (shown in Claim 3.4). However, the steps in the sequence are not mapped to time. Thus, we need a scheme to assign new times to the reordered steps so that the new timed computation does not violate the timing constraints of M. This is the major difference between our lower bound proof for unknown parameter models and that in [3] for the asynchronous model; in the latter no timing scheme is necessary because the asynchronous model imposes no timing constraints.

In the following, we define a retiming scheme that also encompasses the reordering scheme presented above.

Let us first assign a new mapping T'' to every step π in $\beta\gamma$, including all the steps of the network N if M is an MP model, such that $T''(\pi) = T(\pi) \cdot \frac{c}{c_2'}$. That is, every process (except N) takes a step at every time that is a multiple of c. Since in an MP model, the delivery steps of N are retimed along with other steps, the message delay is now changed from d_2' to $d_2' \cdot \frac{c}{c_2'}$. Note that the assignment of T'' does not change the relative ordering of the steps in β because it changes every step time by the same proportion (namely, $\frac{c}{c_2'}$).

We now reorder and assign new times (the mapping T') to every step in (β_k, T''). Intuitively, in order to obtain a timed computation as in Figure 3.1, σ_k and every step that σ_k depends on have to move earlier in time into the first half of (β_k, T''), and τ_k and every step that depends on τ_k have to move later in time into the second half of (β_k, T''). As a result, σ_k will occur before τ_k.

Let $t_k = k \cdot \frac{Bc}{c_2'}$, where $0 \leq k \leq s-1$. Thus, β_k occurs during $[t_{k-1}, t_k)$ under T''. Note that $t_k - t_{k-1} = \frac{Bc}{c_2'}$.

[3]The statement of Theorem 2 in [3] does not explicitly include this result. For conciseness, we do not include a copy of the relevant part of the proof (the middle of pp. 453–455).

1. (Earlier retiming.) Let π be any step in β_k by a process in P that σ_k depends on (in an MP model, this means $proc(\pi) = proc(\sigma_k)$). Retime π such that $T'(\pi) = \frac{t_{k-1}+T''(\pi)}{2}$. The step is moved backward halfway to the beginning of β_k.
2. (Later retiming.) Let σ be any step in β_k by a process in P that depends on τ_k (in an MP model, this means $proc(\sigma) = proc(\tau_k)$). Retime σ such that $T'(\sigma) = \frac{T''(\sigma)+t_k-c}{2}$. The step is moved forward approximately halfway to the end of β_k.
3. (Stationary retiming.) All other steps in β_k and all steps in γ are assigned the same times as in T''.

For all k, $1 \le k \le s-1$, let β''_k be the result of reordering and retiming steps in β_k according to T', and let $\beta'' = \beta''_1\beta''_2 \dots \beta''_{s-1}$.

If M is an MP model, let β' be the result of changing the states of the network in β'' so that in each step of the network, the state of the network is consistent with all the send steps of regular processes and all the deliver steps of the network in β'' that have happened so far ("consistent" means that a delivery step of a message happens after its send step). If M is an SM model, let $\beta' = \beta''$.

In summary, (β, T) is now transformed to (β', T') using (β'', T'') as an intermediate computation. To show the theorem, it suffices to show that (1) β' is a computation leading to the same global state as β (Claim 3.3); (2) β' contains less than s sessions (Claim 3.4); and (3) $(\beta'\gamma, T')$ is admissible for M (Claim 3.5).

CLAIM 3.3. *β' is a computation that leaves the system in the same global state as β does.*

Proof. We first prove that $\le_\beta$ holds in β'.

For any k, $1 \le k \le s-1$, pick any two steps π and π' in β_k such that $\pi \le_\beta \pi'$. Thus $T(\pi) \le T(\pi')$ (recall that T is the original timing). We only need to prove that $T'(\pi) \le T'(\pi')$ because π occurs earlier than π' in T.

Each of π and π' was retimed by one of the earlier, later, or stationary retiming methods. Clearly the desired ordering is preserved

(1) when π stays the same and π' either stays the same or moves later in time; or

(2) when π' stays the same and π either stays the same or moves earlier in time; or

(3) when π and π' both move in the same direction.

The other cases cannot occur, since if π moves later in time, then so does π', and if π' moves earlier in time, then so does π.

Since β' does not violate $\le_\beta$ in both SM and MP models, and all the states of *net* in β' are consistent with the steps of β'' by the definition of β' in MP, it follows that β' is a computation resulting in the same global state as β. □

CLAIM 3.4. *β' contains at most $s-1$ sessions.*

Proof. For all k, $1 \le k \le s-1$, let $h_k = T'(\tau_k)$. We show, by induction on k, that time period $[0, h_k)$ in (β', T') contains at most $k-1$ sessions.

For the basis, $[0, h_1)$ contains no session because it does not contain any port event for y_0.

Assume, by way of induction, that $[0, h_{k-1})$ contains at most $k-2$ sessions. We prove that $[0, h_k)$ contains at most $k-1$ sessions. It is clear from the construction that $(h_{k-1}, t_{k-1}]$ contains no session because it does not contain any port event of y_{k-1} and, similarly, $[t_{k-1}, h_k)$ does not contain any session because it contains no port event of y_{k-1}. Since the port process of y_{k-1} takes only one step at h_{k-1}, it is clear

that $[h_{k-1}, h_k)$ contains at most one session. Therefore, $[0, h_k)$ contains at most $k-1$ sessions.

A similar argument shows that $[h_{s-1}, t_{s-1}]$ contains at most one session. Since $[0, h_{s-1})$ contains at most $s-2$ sessions, it follows that there are at most $s-1$ sessions in $[0, t_{s-1}]$. □

CLAIM 3.5. *$(\beta'\gamma, T')$ is a timed computation that is admissible for M.*

Proof. By Claim 3.3, β' is a computation that leads to the same global state as β. Therefore, $\beta'\gamma$ is also a computation because $\beta\gamma$ is a computation. Since γ is infinite, so is $\beta'\gamma$. It remains to show that the timing T' conforms to the timing constraints of model M.

Constraints on step time. Let π_i and π_{i+1} be consecutive steps of a single process in β_k for some k. Then, for some $\Delta \geq 0$, it is true that $T''(\pi_i) = t_{k-1} + \Delta$ and $T''(\pi_{i+1}) = t_{k-1} + \Delta + c$.

The distance between the two steps is minimized when both steps are retimed by the same retiming method because it is not possible that π_i is retimed by the later retiming while π_{i+1} is retimed by the earlier retiming. If both π_i and π_{i+1} are retimed by the same retiming method (either later or earlier retiming), $T'(\pi_{i+1}) - T'(\pi_i)$ is equal to $c/2$.

None of the retiming methods retimes a step outside its original segment: If a step is in segment β_k, after the retiming, it is still in segment β'_k. Thus, neither π_i nor π_{i+1} is retimed outside the segment. Since the maximum time elapsed between π_i and π_{i+1} in β_k is B, and in T'', all times are shrunk in proportion to $\frac{c}{c'_2}$, the distance between the two steps can never be larger than $t_k - t_{k-1} \leq B\frac{c}{c'_2}$.

For the timing of steps in different segments, let π_j and π_{j+1} be the consecutive steps of a process, each of which is in a different segment, say, β_k and β_{k+1}, respectively. Then $T''(\pi_j) = t_k$ and $T''(\pi_{j+1}) = t_k + c$ (this is because B is divisible by c'_2). Since the two steps are in different segments, the distance between the two steps is minimized when π_j is retimed by the later retiming while π_{j+1} is retimed by the earlier retiming. However, neither of these retimings results in a change, i.e., $T'(\pi_j) = T''(\pi_j)$ and $T'(\pi_{j+1}) = T''(\pi_{j+1})$. Therefore, $T'(\pi_{i+1}) - T'(\pi_i) = c$. Since the two steps are in different segments, the distance between the two steps is maximized when π_j is retimed by the earlier retiming while π_{j+1} is retimed by the later retiming. In this case, π_j moves to $(t_{k-1} + t_k - c)/2$ and π_{j+1} moves to $(t_k + t_{k+1} - c)/2$. Therefore, the distance between the two steps cannot be larger than $B\frac{c}{c'_2}$ because $T'(\pi_{i+1}) - T'(\pi_i) = \frac{t_{k+1} - t_{k-1}}{2} \leq B\frac{c}{c'_2}$.

We first check the minimum step time.

- If c_1 is unknown, then there exists a positive constant (namely $c/2$) which is the minimum step time in (β', T').
- If c_1 is known, by condition SC2 (or MC2), the minimum step $(c/2)$ in (β', T') is bigger than or equal to the minimum step time of M.
- If c_2 is known, the minimum step $(c/2)$ is less than or equal to c_2 because (1) by SC3 (MC3), $c'_2 = c_2$ and $B\frac{c}{c_2} \leq c_2$; and (2) by the assumption for Case 2 of the main proof, $B > c_2$.

Now we check the maximum step time.

- If c_2 is unknown, then there exists a positive constant (namely $B\frac{c}{c'_2}$) which is the maximum step time in (β', T').
- If c_2 is known, the maximum step time $(B\frac{c}{c'_2})$ in (β', T') is less than or equal to c_2 because of SC3 and MC3.

- If c_1 is known, the maximum step time ($B\frac{c}{c_2'}$) in (β', T') is bigger than or equal to c_1 because (1) by the assumption for Case 2, $B > c_2'$ and thus $B\frac{c}{c_2'} > c$; and (2) by SC2 and MC2, $c \geq c_1$.

Constraints on message delay. The steps that move the farthest due to the earlier retiming from T'' to T' are those at the end of each segment because they do not move outside their segments. Let π_{last} be the step at the end of β_k. $T''(\pi_{last}) = t_k - c$. By the earlier retiming, $T'(\pi_{last}) = (t_{k-1} + t_k - c)/2 = t_k - \frac{1}{2}B\frac{c}{c_2'} - c/2$.

The steps that move the farthest due to the later retiming from T'' to T' are those at the start of each segment because they do not move outside their segments. Let π_{start} be the step at the start of β_k. $T''(\pi_{start}) = t_{k-1}$. By the later retiming, $T'(\pi_{start}) = (t_{k-1} + t_k - c)/2 = t_k - \frac{1}{2}B\frac{c}{c_2'} - c/2$. Therefore, the maximum distance that a step can move from T'' to T' is bounded by $\frac{1}{2}B\frac{c}{c_2'} - c/2$.

Let π_s and π_r be the send and receive steps of any message in β. Recall that under T'', all the message delays are $d_2'\frac{c}{c_2'}$. Thus, $T''(\pi_r) - T''(\pi_s) = d_2'\frac{c}{c_2'}$. Let $d' = d_2'\frac{c}{c_2'}$. Because both π_r and π_s can move $\frac{1}{2}(B\frac{c}{c_2'} - c)$ time from T' to T'', each in opposite directions, $d' - B\frac{c}{c_2} + c \leq T'(\pi_r) - T'(\pi_s) \leq d' + B\frac{c}{c_2} - c$.

We first check the minimum message delay.

- If d_1 is unknown, then there exists a positive constant that is the minimum message delay in (β', T') because (1) $T'(\pi_r) - T'(\pi_s) \geq d' - B\frac{c}{c_2} + c$; (2) by MC1, $d' - B\frac{c}{c_2'} = (d_2' - B)\frac{c}{c_2'} \geq 0$; and (3) c is a positive real.
- If d_1 is known, the minimum message delay $(d' - B\frac{c}{c_2'} + c)$ in (β', T') is bigger than or equal to d_1 because of MC4.

We now check the maximum message delay.

- If d_2 is unknown, then there exists a positive constant (namely $d' + B\frac{c}{c_2'} - c$) that is the maximum message delay in (β', T') because $d' + B\frac{c}{c_2'} - c \geq T'(\pi_r) - T'(\pi_s) > 0$.
- If d_2 is known, the maximum message delay $(d' + B\frac{c}{c_2'} - c)$ is less than or equal to d_2 because of MC5. □

To finish the proof of the main theorem, $(\beta'\gamma, T')$ is an infinite timed computation admissible for M but with fewer than s sessions, by Claims 3.4 and 3.5. This contradicts the assumed correctness of A. □

4. Algorithmic counting methods. We develop five methods to count the number of sessions during a computation (cf. Table 4.1). These methods differ in the ways they use the known timing information of a model to count sessions. An (s, n)-session algorithm can be obtained for a model simply by combining all the applicable methods to the model without increasing the asymptotic time complexity of any of those methods. This can be done by running each method "side by side," halting when the first of them finishes [5]. Since there is only a constant number of methods running at the same time, the combination does not affect the asymptotic time complexity of the algorithm. The resulting upper bound on the time needed to solve the session problem in a model is the minimum of the time complexity of all the methods applicable to the model.

We now describe each of the counting methods. A message is denoted $m(i, j, k)$, where i is the identifier of the sending process p_i, j is an integer in $[0, s-1]$, and k is an integer. We let $*$ be a "don't care" value. The port for a port process i is denoted y_i. In an MP model, y_i denotes process i's buffer of incoming messages.

TABLE 4.1
Counting methods.

Technique	Timing information	Running time
Explicit communication (EC)	None	$(s-1)\cdot d_2 + c_2$ or $(s-1)\cdot c_2\Theta(\log n) + c_2$
Step time (ST)	c_1, c_2	$(s-1)\cdot \frac{c_2}{c_1}\cdot c_2 + c_2$
Combination 1 (CB1)	c_1, d_1, d_2	$(s-2)\cdot(\frac{c_2}{c_1}\cdot u + u + 2c_2) + d_2 + 3c_2$
Message delay (MD)	d_1, d_2	$(s-2)(\frac{d_2}{d_1}\cdot u + u + 2c_2) + d_2 + 2c_2$
Combination 2 (CB2)	c_2, d_1	$(s-1)\cdot c_2 \cdot \frac{d_2+c_2}{d_1} + c_2$

In describing the methods, we use a subroutine called *broadcast* as a generic operator for communication. In MP models, *broadcast* is accomplished by having each process send a message to all the processes, including itself. In specifying the algorithm for counting, we use a message format with three fields, $m(i, j, k)$, where i indicates the identifier of the process sending the message; j is the session number that process i is currently in; and k is an integer used as a local variable in the algorithm. When marked with $*$, the field denotes "don't care" (i.e., a message with any value on that field).

We now explain how to achieve a broadcast in an SM model. Recall that at most a processes can access any specific shared variable. We conceptually organize the processes and shared variables into a tree with $\Theta(\log n)$ levels. In order for a port process to broadcast information to all other port processes, the information travels up the tree to the root and then down from the root to all the leaves. See Appendix A for more details.

4.1. Explicit communication (EC). The EC method (see Figure 4.1), originally presented and analyzed in [3] for the asynchronous SM model, and in [5] for the asynchronous MP model, does not require any timing information to solve the (s, n)-session problem. It can be used in any timing model because the correctness of the method does not depend on specific step time or message delay.

```
session := 0; msgs := ∅;
while ( session < s − 1 )
    msgs := msgs ∪ y_i; /* port event; recall y_i = buf_i in MP */
    if for all j ∈ {1, ..., n}, m(j, session, *) is in msgs
    then
        session := session + 1;
    end if;
    broadcast m(i, session, *); /* port event */
end while;
Enter an idle state.
```

FIG. 4.1. *Technique EC for process i.*

THEOREM 4.1. *EC solves the (s, n)-session problem in time $(s-1)\cdot c_2\cdot\Theta(\log n)+c_2$ in an SM model and in time $(s-1)\cdot d_2 + c_2$ in an MP model.*

The basic intuition for the method is that since it does not use any timing information, a process relies only on communication with other processes at every session to recognize that there is one session. Each process executes one port event, broad-

```
B := c2/c1;
count := session := 0;
while ( session < s − 1 )
    if (count ≥ B)
    then
        count := 0;
        session := session + 1;
    end if;
    count := count + 1;
    access y_i; /* port event; recall y_i = buf_i in MP */
end while;
Enter an idle state.
```

FIG. 4.2. *Technique ST for process i.*

casts the fact to every process at each step, and repeats this step until it hears that every process has executed another port event. Then it increments *session*. It performs these steps $s-1$ times. Then, after it executes one additional port event, it enters an idle state.

4.2. Step time (ST). The ST method (see Figure 4.2), originally presented and analyzed in [5] for the semisynchronous model, requires information about the maximum and minimum step times (c_2 and c_1). For convenience of presentation, we assume that c_2 is divisible by c_1. This method can also be used for both an SM model and an MP model.

In this method, processes use timing information about relative step times to determine when a session occurs. Each process executes $\frac{c_2}{c_1}$ port events. During this interval, at least $\frac{c_2}{c_1} \cdot c_1 = c_2$ time elapses, since c_1 is the minimum step time. Since every process performs at least one port event within time c_2 (since c_2 is the maximum step time), at least one session has occurred by the time that the process finishes $\frac{c_2}{c_1}$ port events. Each process repeats the above procedure $s-1$ times. After it executes one additional port event, it enters an idle state.

THEOREM 4.2. *ST solves the (s,n)-session problem in time $(s-1) \cdot \frac{c_2}{c_1} c_2 + c_2$ in both an SM model and an MP model.*

4.3. Combination 1 (CB1). The CB1 method (see Figure 4.3) requires information about the minimum step time (c_1) and the minimum and maximum message delays (d_1 and d_2). For convenience of presentation, we assume that $u = d_2 - d_1$ is divisible by c_1.

The correctness of this method relies on the following observation. If a process p_i receives a message m from a process p_j at time t, then the message must have been sent no later than $t - d_1$, because it takes at least d_1 time for a message to be delivered. All the messages received by p_i after $t + d_2 - d_1$ must have been sent after m was, because it takes at most d_2 time for a message to be delivered. Based on this idea, each process broadcasts a message at every step. (1) When a process initially receives one message from every process, it recognizes there is one session because each process must have taken one step to send the message. (2) Then when it receives another set of messages from every process after time $u = d_2 - d_1$, there must have been another session after the initial session because the second set of messages must have been sent after the first session occurred. Time u can be measured by counting

```
B := u/c1; /* u = d2 − d1 */
count := session := 0;
msgs := ∅;
while (session < s − 1)
    msgs := msgs ∪ buf_i; /* port event */
    if (session = 0) or (count ≥ B)
    then
        if for all j ∈ {1, ..., n}, m(j, ∗, ∗) is in msgs
        then /∗ condition 1 ∗/
            count := 0;
            session := session + 1;
            msgs := ∅;
        end if;
    end if;
    broadcast m(i, ∗, ∗); /* port event */
    count := count + 1;
end while;
Enter an idle state.
```

FIG. 4.3. *Technique CB*1 *for process i.*

steps, using the known minimum step time (c_1) (i.e., when a process takes $\frac{u}{c_1}$ local steps, at least u time is guaranteed to be elapsed).

Each process performs the second procedure $s-2$ times. Then it enters an idle state after taking an additional step. The running time of CB1 is $(\frac{c_2}{c_1}u+u+2c_2)(s-2)+d_2+3c_2$, as we now explain. It takes at most $\frac{u}{c_1}\cdot c_2$ time to count $\frac{u}{c_1}$ steps, at most $u+2c_2$ time for a process to receive another set of message after it recognizes there was a session, at most d_2+2c_2 time to receive the initial set of messages, and finally one more step to accomplish the last session. The detailed proof of the following theorem can be found in Appendix A.

THEOREM 4.3. *CB*1 *solves the* (s,n)*-session problem within time* $(s-2)\cdot(\frac{c_2}{c_1}u+u+2c_2)+d_2+3c_2$ *if* c_1*,* d_1*, and* d_2 *are known.*

4.4. Message delay (MD). The MD method (see Figure 4.4) requires information about the lower bound d_1 and upper bound d_2 on message delay. MD differs from CB1 only in one way: a process recognizes that time u has elapsed by counting the number of times that a certain message is being passed between two processes, using the known minimum message delay. For example, when a process p_i broadcasts a message at time t, the message is received by p_j no earlier than time $t+d_1$. So if the message is passed between them (or any process because of the minimum message delay) more than u/d_1 times, then we know at least u time has elapsed. The running time of MD is equal to that of CB1 with the $\frac{c_2}{c_1}$ factor replaced by $\frac{d_2}{d_1}$.

The detailed proof of the following theorem can be found in Appendix A.

THEOREM 4.4. *MD solves the* (s,n)*-session problem within time* $(s-2)\cdot(\frac{d_2}{d_1}u+u+2c_2)+d_2+3c_2$ *if* d_1 *and* d_2 *are known.*

4.5. Combination 2 (CB2). The CB2 method (see Figure 4.5) can be used if the maximum step time c_2 and the minimum message delay d_1 are known. The known minimum message delay (d_1) can be used to measure the elapsed time between the send time of a message and the receive time of the same message. We know at

$B := \frac{u}{d_1}$;
$count := session := 0$;
$msgs := \emptyset$;
while($session < s - 1$)
  $msgs := msgs \cup buf_i$;
  if $(session = 0)$ or $(count \geq B)$
  then
    if for all $j \in [n]$, $m(j, *, *)$ is in $msgs$
    then /* condition 1 */
      $count := 0$;
      $session := session + 1$;
      $msgs := \emptyset$;
    end if;
  end if;
  if there is any $m(*, session, *)$ in $msgs$
  then $count := \max\{count, k : m(*, session, k) \in msgs\ \}$;
  broadcast $m(i, session, count + 1)$;
end while;
Enter an idle state.

FIG. 4.4. *Technique MD for process i.*

$count := 0$;
$msgs := \emptyset$;
while($count < s - 1$)
  $msgs := msgs \cup buf_i$;
  $count := \max\{k, count : m(*, *, k) \in msgs\ \}$;
  broadcast $m(i, *, count + \frac{c_2}{d_1})$;
end while;
Enter an idle state.

FIG. 4.5. *Technique CB2 for process i.*

least d_1 time has passed between the send and receive. In addition, because of the known maximum step time, it is possible to estimate how many steps a process takes within time d_1 (at least $\frac{d_1}{c_2}$ steps). Therefore, a process can deduce that if it receives a message sent after the last session, there have been at least $\frac{d_1}{c_2}$ sessions after that last session.

We can inductively apply the above argument starting from session 0. Initially, a process starts by sending a message to all, and as soon as it receives a message from all other processes, it knows that there are at least $\frac{d_1}{c_2}$ sessions in the computation. It increments its counter by $\frac{d_1}{c_2}$ and sends another message piggybacking the value of that counter. If it receives a message with a counter x, it knows that there are at least $x + \frac{d_1}{c_2}$ sessions. Then it updates its counter to $x + \frac{d_1}{c_2}$ and sends another message with the value of that counter. It continues the above until its counter is larger than $s - 1$. Then it takes one more step and enters an idle state.

The detailed proof of the following theorem can be found in Appendix A.

THEOREM 4.5. *Technique CB2 solves the* (s, n)*-session problem in time* $(s-1) \cdot \frac{d_2+c_2}{d_1} c_2 + c_2$ *if* c_2 *and* d_1 *are known.*

5. Shared memory results. In this section, we show that the upper bounds we presented in section 4 for the SM models are asymptotically tight by obtaining the matching lower bounds. We use Theorem 3.1 to obtain the lower bounds. To prove a given lower bound for an SM model, we simply check whether there exist c, c'_1, and c'_2 that satisfy SC1, SC2, and SC3.

5.1. Counting with EC. Suppose that only EC is used. The resulting upper bound is $(s-1) \cdot c_2 \cdot \Theta(\log n)$.

We now show that this bound is asymptotically tight. In particular, we show that if either c_1 or c_2 (or both) is unknown, then the lower bound is $(s-1) \cdot c_2 \cdot \log_a n$.

COROLLARY 5.1. *Let $c_2 \in \mathcal{R}^+$. For $\mathcal{M}[?, c_2]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1) \cdot c_2 \cdot \log_a n$.*

Proof. Let $c = c_2 / \log_a n$. Let c'_1 be some constant less than or equal to c_2. Let $c'_2 = c_2$. Let $f(x,y) = y \cdot \log_a n$.

As $c'_1 \leq c_2$ and $B = f(c'_1, c'_2) = c'_2 \cdot \log_a n$, SC1 is satisfied. As $B \frac{c}{c'_2} = c'_2$, SC3 is satisfied. As c_1 is unknown, we do not consider SC2. □

COROLLARY 5.2. *Let $c_1 \in \mathcal{R}^+$. For $\mathcal{M}[c_1, ?]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1) \cdot c_2 \cdot \log_a n$.*

Proof. Let $c = 2 \cdot c_1$. Let $c'_1 = c_1$. Let c'_2 be some constant greater than or equal to c_1. Let $f(x,y) = y \cdot \log_a n$.

As $B = f(c'_1, c'_2) = c'_2 \cdot \log_a n$, SC1 is satisfied. As $\frac{1}{2}c = c_1$ and $c'_1 = c_1$, SC2 is satisfied. As c_2 is unknown, we do not consider SC3. □

Each of Corollaries 5.1 and 5.2 separately implies that a lower bound for $\mathcal{M}[?, ?]$ is $(s-1) \cdot c_2 \cdot \log_a n$.

5.2. Counting with EC and ST bounds. If processes know c_1 and c_2, then they can use both methods ST and EC in order to count. This results in an algorithm with running time $(s-1) \cdot c_2 \cdot \min\{\frac{c_2}{2c_1}, \Theta(\log n)\} + c_2$.

We now show that this bound is asymptotically tight.

COROLLARY 5.3. *Let $c_1, c_2 \in \mathcal{R}^+$. For $\mathcal{M}[c_1, c_2]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1) \cdot c_2 \cdot \min\{\frac{c_2}{2c_1}, \log_a n\}$.*

Proof. Let $c = 2c_1$. Let $c'_1 = c_1$. Let $c'_2 = c_2$. Let $f(x,y) = y \cdot \min\{\frac{y}{x}, \log_a n\}$.

SC1 holds because $B = f(c'_1, c'_2) = c'_2 \cdot \min\{\frac{c'_2}{2c'_1}, \log_a n\} \leq c'_2 \cdot \log_a n$. SC2 holds because $c'_1 = c_1 \leq \frac{c}{2}$. SC3 holds because $B \frac{c}{c'_2} = f(c'_1, c'_2) \cdot \frac{c}{c'_2} = c'_2 \cdot \min\{\frac{c'_2}{2c'_1}, \log_a n\} \frac{2c_1}{c'_2} \leq c'_2$. □

6. Message passing results. In this section, we show that the upper bounds we presented in section 4 for the MP models are asymptotically tight by obtaining the matching lower bounds. We use Theorem 3.1 to obtain the lower bounds. To prove a given lower bound for an MP model, we simply check whether there exist c, c'_1, c'_2, d'_1, and d'_2 that satisfy MC1 through MC5.

6.1. Counting with EC. The use of EC alone in an MP model gives an upper bound of $(s-1) \cdot d_2 + c_2$.

We show that if no other method can be used, then this bound is asymptotically tight. The models that allow the use of EC only are models 0, 1, 2, 4, 5, 8, 9, and 10. Using corollaries to Theorem 3.1, we show that the lower bounds for models 5, 9, and 10 are $(s-1) \cdot d_2$. The result for model 5 implies the same lower bound for models 1 and 4 (since 1 and 4 have less timing information than model 5). The result for model 10 implies the same lower bound for models 2 and 8. The result for model 1 implies the same lower bound for model 0.

First, we give the corollary for model 5.

COROLLARY 6.1. *Let c_2 and d_2 be positive reals. For $\mathcal{M}[?, c_2, ?, d_2]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1)\cdot d_2$.*

Proof. Let $c = \min\{\frac{c_2^2}{d_2}, \frac{c_2}{2}\}$. Let c_1' be some constant less than or equal to c_2. Let $c_2' = c_2$. Let d_1' be some constant less than or equal to d_2. Let $d_2' = d_2$. Let $f(w,x,y,z) = z$.

MC1 is satisfied by the choice of c_1', c_2', d_1', and d_2' and because $B = f(c_1', c_2', d_1', d_2') = d_2' = d_2$.

MC3 is satisfied because $B \cdot \frac{c}{c_2'} \le d_2 \cdot \frac{c_2^2/d_2}{c_2'} \le c_2$.

MC5 is satisfied because $(d_2' + B)\frac{c}{c_2'} - c \le 2d_2 \cdot \frac{c_2/2}{c_2} - c < d_2$.

MC2 and MC4 are not considered because c_1 and d_1 are unknown. □

Next we give the corollary for model 9.

COROLLARY 6.2. *Let c_1 and d_2 be positive reals. For $\mathcal{M}[c_1, ?, ?, d_2]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1)d_2$.*

Proof. Let $c = 2c_1$. Let $c_1' = c_1$. Let $c_2' = 4c_1$. Let d_1' be some constant less than or equal to d_2. Let $d_2' = d_2$. Let $f(w,x,y,z) = z$.

MC1 is satisfied because $B = f(c_1', c_2', d_1', d_2') = d_2' = d_2$.

MC2 is satisfied because $\frac{c}{2} = c_1$.

MC5 is satisfied because $(d_2' + B)\frac{c}{c_2'} - c \le 2d_2\frac{2c_1}{4c_1} = d_2$.

MC3 and MC4 are not considered because c_2 and d_1 are unknown. □

Finally, we give the corollary for model 10.

COROLLARY 6.3. *Let c_1 and d_1 be positive reals. For $\mathcal{M}[c_1, ?, d_1, ?]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1)\cdot d_2$.*

Proof. Let $c = \max\{2c_1, d_1\}$. Let $c_1' = c_1$. Let c_2' be some constant bigger than or equal to c_1. Let $d_1' = d_1$. Let d_2' be some constant bigger than or equal to d_1. Let $f(w,x,y,z) = z$.

MC1 is satisfied because $B = f(c_1', c_2', d_1', d_2') = d_2'$.

MC2 is satisfied because $\frac{c}{2} \ge \frac{2c_1}{2} = c_1$.

MC4 is satisfied because $(d_2' - B)\frac{c}{c_2'} + c = c \ge d_1 = d_1'$.

MC3 and MC5 are not considered because c_2 and d_2 are unknown. □

6.2. Counting with EC and ST bounds. If both methods EC and ST can be used, the resulting algorithm gives an upper bound of $(s-1)\cdot\min\{d_2, \frac{c_2}{c_1}\cdot c_2\} + c_2$.

We show that this bound is asymptotically tight if no other methods can be used and the following is true:

- $c_2 \le d_2$.

The models that allow the use of EC and ST alone are models 12 and 13. We prove a corollary to Theorem 3.1 for model 13, which implies the same lower bound for model 12.

COROLLARY 6.4. *Let c_1, c_2, and d_2 be positive reals such that $c_2 \le d_2$. For $\mathcal{M}[c_1, c_2, ?, d_2]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1)\cdot\min\{\frac{c_2}{2c_1}c_2, d_2\}$.*

Proof. Let $c = 2c_1$. Let $c_1' = c_1$. Let $c_2' = c_2$. Let d_1' be some constant less than or equal to d_2. Let $d_2' = d_2$. Let $f(w,x,y,z) = \min\{\frac{x}{2w}x, z\}$.

MC1 is satisfied because $B = f(c_1', c_2', d_1', d_2') = \min\{\frac{c_2^2}{2c_1}, d_2\} \le d_2$.

MC2 is satisfied because $\frac{c}{2} = c_1 = c_1'$.

MC3 is satisfied because $B \cdot \frac{c}{c_2'} = \min\{\frac{c_2^2}{2c_1}, d_2\} \cdot \frac{2c_1}{c_2} \le c_2$.

For MC5, we need to show that $(d_2' + B)\frac{c}{c_2'} - c \leq d_2$. Then it suffices to prove that $d_2 - ((d_2' + B)\frac{c}{c_2'} - c) \geq 0$.

$d_2 - ((d_2' + B)\frac{c}{c_2'} - c) \geq d_2 - ((d_2 + \frac{c_2^2}{2c_1})\frac{2c_1}{c_2} - 2c_1) = d_2(1 - \frac{2c_1}{c_2}) - (c_2 - 2c_1) \geq 0$ since $c_2 \leq d_2$.

MC4 is not considered because d_1 is unknown. □

6.3. Counting with EC and MD. Suppose methods EC and MD are used. The only model that can use these two methods alone is model 3 ($\mathcal{M}[?, ?, d_1, d_2]$). The resulting asymptotic upper bound on the per-session cost is $\min\{d_2, \frac{d_2}{d_1}u + 2c_2\}$, where $u = d_2 - d_1$.

We now argue that this bound is asymptotically tight if no other method can be used. First note that if the $2c_2$ term in the MD cost dominates the $\frac{d_2}{d_1}u$ term, then the upper bound is $\Theta(c_2)$ per session, which is obviously tight. Thus we ignore the $2c_2$ term.

Corollary 6.8 in section 6.7 below shows that the lower bound for model 11 ($\mathcal{M}[c_1, ?, d_1, d_2]$) is $(s-1) \cdot \frac{d_2}{d_2+d_1}u$. Since model 3 is weaker than model 11, the same lower bound holds for model 3.

(Case 1) Suppose $d_2 \leq \frac{d_2}{d_1}u$. Then $\frac{d_2}{d_2+d_1}u \geq \frac{d_2}{3}$. Thus the asymptotic per-session upper bound for model 3 is d_2 and the lower bound is $\frac{d_2}{3}$.

(Case 2) Suppose $\frac{d_2}{d_1}u < d_2$. Then $\frac{d_2}{d_2+d_1}u \geq \frac{d_2}{3d_1}u$. Thus the asymptotic per-session upper bound for model 3 is $\frac{d_2}{d_1}u$ and the lower bound is $\frac{d_2}{3d_1}u$.

6.4. Counting with EC and CB2. If methods EC and CB2 are used, the resulting asymptotic upper bound on the per-session cost is $\min\{d_2, \frac{d_2+c_2}{d_1} \cdot c_2\}$. The only model that can use these two methods alone is model 6 ($\mathcal{M}[?, c_2, d_1, ?]$).

We now show this bound is asymptotically tight if

- $2c_2 \leq 3d_1$.

Under this assumption, the CB2 term, $\frac{d_2+c_2}{d_1} \cdot c_2$, is at most $\frac{5d_2}{2d_1} \cdot c_2$, which is at most $\frac{15}{4}d_2$.

COROLLARY 6.5. *Let c_2 and d_1 be positive reals such that $c_2 \leq \frac{3}{2}d_1$. For $\mathcal{M}[?, c_2, d_1, ?]$, there exists no algorithm that solves the (s, n)-session problem within time less than $(s-1) \cdot \frac{2d_2}{3d_1} \cdot c_2$.*

Proof. Let c_1' be some constant less than or equal to c_2. Let c_2' equal c_2. Let d_1' equal d_1. Let d_2' be some constant bigger than d_1. Let $c = \frac{3d_1}{2d_2'} \cdot c_2$. Let $f(w, x, y, z) = \frac{2z}{3y} \cdot x$.

MC1 holds because $B = f(c_1', c_2', d_1', d_2') = \frac{2d_2'}{3d_1'} \cdot c_2'$. By the assumption that $c_2 \leq \frac{3}{2}d_1$, B is less than or equal to d_2'.

MC3 holds because $B \cdot \frac{c}{c_2'} = c_2'$.

MC4 holds because $c_2 \leq \frac{3}{2}d_1$, and $(d_2' - B) \cdot \frac{c}{c_2'} + c = 3d_1 - c_2 + c \geq 3d_1 - 3/2d_1 + c > d_1 = d_1'$.

MC2 and MC5 are not considered because c_1 and d_2 are unknown. □

6.5. Counting with EC, ST, and CB2. If methods EC, ST, and CB2 are used, the resulting asymptotic upper bound on the per-session cost is $\min\{d_2, \frac{c_2}{c_1} \cdot c_2, \frac{d_2+c_2}{d_1} \cdot c_2\}$. The only model in which exactly these methods can be used is model 14 ($\mathcal{M}[c_1, c_2, d_1, ?]$).

We now argue that this bound is asymptotically tight if no other method can be used, assuming

- $2c_2 \leq d_1$.

Corollary 6.6 shows that the lower bound for model 14 is $(s-1)\cdot\min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2\}$, assuming $2c_2 \le d_1$. We show that this bound is asymptotically tight.

Because $2c_2 \le d_1$, $\frac{2d_2}{3d_1}c_2 \le d_2$, and $\frac{d_2+c_2}{d_1} \le \frac{3d_2}{2d_1}$. Thus the per-session lower bound is $\min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2\}$, and the per-session upper bound is $\min\{d_2, \frac{c_2}{c_1}c_2, \frac{3d_2}{2d_1}c_2\} = \min\{\frac{c_2}{c_1}c_2, \frac{3d_2}{2d_1}c_2\}$.

COROLLARY 6.6. *Let c_2 and d_1 be positive reals such that $2c_2 \le d_1$. For $\mathcal{M}[c_1, c_2, d_1, ?]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1)\cdot\min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2\}$.*

Proof. By the hypothesis of the corollary,

$$d_1 \ge 2c_2. \tag{6.1}$$

Let $c_1' = c_1$. Let $c_2' = c_2$. Let $d_1' = d_1$. Let d_2' be some real number bigger than $\frac{c_2}{2c_1}c_2$. Let $c = \max\{2c_1, \frac{3d_1}{2d_2'}c_2\}$.

Let $f(w,x,y,z) = \min\{\frac{x}{2w}x, \frac{2z}{3y}x\}$.

We show that they satisfy the hypothesis of Theorem 3.1. MC5 is not considered since d_2 is unknown.

Note that $B = f(c_1', c_2', d_1', d_2') = \min\{\frac{c_2}{2c_1}c_2, \frac{2d_2'}{3d_1}c_2\}$.

Case 1. Suppose $\frac{c_2}{2c_1}c_2 < \frac{2d_2}{3d_1}c_2$. Then

$$\frac{2c_1}{c_2} > \frac{3d_1}{2d_2'}, \tag{6.2}$$

and thus $c = 2c_1$. For MC1, clearly $B = \frac{c_2}{2c_1}c_2 < d_2'$ by the definition of d_2'.

For MC2, $\frac{c}{2} = c_1$.

For MC3, $B\frac{c}{c_2'} = c_2$.

For MC4,

$$\begin{aligned}(d_2' - B)\tfrac{c}{c_2'} + c &= d_2'\tfrac{2c_1}{c_2} - c_2 + 2c_1 \\ &\ge d_2'\tfrac{3d_1}{2d_2'} - c_2 && \text{because of (6.2)} \\ &\ge \tfrac{3d_1}{2} - \tfrac{d_1}{2} && \text{because of (6.1)} \\ &= d_1.\end{aligned}$$

Case 2. Suppose $\frac{c_2}{2c_1}c_2 \ge \frac{2d_2'}{3d_1}c_2$. Then

$$\frac{2c_1}{c_2} \le \frac{3d_1}{2d_2'}, \tag{6.3}$$

and thus $c = \frac{3d_1}{2d_2'}\cdot c_2$. For MC1, $B = \frac{2d_2'}{3d_1}\frac{d_1}{2} < d_2'$ because of (6.1).

For MC2, $\frac{c}{2} = \frac{1}{2}\frac{3d_1}{2d_2'}c_2 \ge c_1$ because of (6.3).

For MC3,

$$B\frac{c}{c_2'} \le \frac{2d_2'}{3d_1}c_2\frac{\frac{3d_1}{2d_2'}c_2}{c_2} = c_2.$$

For MC4,

$$\begin{aligned}(d_2' - B)\tfrac{c}{c_2'} + c &= (d_2' - \tfrac{2d_2'}{3d_1}c_2)\tfrac{c}{c_2'} + c \\ &= \tfrac{3d_1}{2} - c_2 + c \\ &> \tfrac{3d_1}{2} - c_2 \\ &\ge \tfrac{3d_1}{2} - \tfrac{d_1}{2} && \text{because of (6.1)} \\ &= d_1. \qquad \square\end{aligned}$$

6.6. Counting with EC, MD, and CB2. If methods EC, MD, and CB2 are used, the resulting asymptotic upper bound on the per-session cost is $\min\{d_2, \frac{d_2}{d_1} \cdot u + 2c_2, \frac{d_2+c_2}{d_1} \cdot c_2\}$. The only model in which exactly these methods can be used is model 7 ($\mathcal{M}[?, c_2, d_1, d_2]$).

We now show this upper bound is asymptotically tight if

- $2c_2 \leq d_1$, and
- $d_2 \leq \frac{3}{2}d_1$ or $\frac{3}{2}d_1 + c_2 \leq d_2$.

As in section 6.3, we ignore the $2c_2$ term in the MD expression.

Corollary 6.7 proves that the per-session lower bound is $(s-1)\min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$. We show that this bound is asymptotically tight.

(Case 1) If $\frac{d_2}{d_1}u \leq \min\{d_2, \frac{d_2+c_2}{d_1}c_2\}$, then the upper bound is $\Theta(\frac{d_2}{d_1}u)$. The lower bound is $\frac{d_1}{3d_2}u$ because $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_1}{3d_2}u\}$ (cf. section 6.3); and since $c_2 \leq \frac{1}{2}d_1$, $\frac{d_2}{d_1}u \leq \frac{d_2+c_2}{d_1}c_2 \leq \frac{3d_2}{2d_1}c_2$, and thus $\frac{d_2}{3d_1}u < \frac{2d_2}{3d_1}c_2$.

(Case 2) If $\frac{d_2+c_2}{d_1}c_2 \leq \min\{d_2, \frac{d_2}{d_1}u\}$, then $\frac{d_2+c_2}{d_1}c_2 \leq \frac{3d_2}{2d_1}c_2$ because $c_2 \leq \frac{1}{2}d_1$. Thus the per-session upper bound is $\Theta(\frac{d_2}{d_1}c_2)$. The lower bound is $\frac{d_1}{3d_2}c_2$ because $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_1}{3d_2}u\}$, and because $\frac{d_2}{d_1}u \geq \frac{d_2+c_2}{d_1}c_2 > \frac{d_2}{d_1}c_2$, $\frac{d_2}{3d_1}u > \frac{d_2}{3d_1}c_2$.

(Case 3) If $d_2 \leq \min\{\frac{d_2}{d_1}u, \frac{d_2+c_2}{d_1}c_2\}$, then the upper bound is $\Theta(d_2)$. The lower bound is $\frac{1}{3}d_2$ because $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_1}{3d_2}u\}$, and since $c_2 \leq \frac{1}{2}d_1$, $d_2 \leq \frac{d_2+c_2}{d_1}c_2 \leq \frac{3d_2}{2d_1}c_2$, and thus $\frac{1}{3}d_2 < \frac{2d_2}{3d_1}c_2$.

COROLLARY 6.7. *Let c_2, d_1, and d_2 be positive reals such that $2c_2 \leq d_1$ and either $d_2 \leq \frac{3}{2}d_1$ or $\frac{3}{2}d_1 + c_2 \leq d_2$. For $\mathcal{M}[?, c_2, d_1, d_2]$, there exists no algorithm that solves the (s, n)-session problem within time less than $(s-1)\min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$,*

Proof.

Case 1. $d_2 \leq \frac{3}{2}d_1$.

Then $\min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\} \leq \frac{2d_2}{3d_1}c_2 = c_2$. The lower bound holds because a process has to take at least s steps to solve the (s, n)-session problem.

Case 2. $d_2 \geq \frac{3}{2} \cdot d_1 + c_2$ and $c_2 \leq \frac{3}{2}d_1$.

Let $c = \frac{3d_1}{2d_2}c_2$. Let c_1' be some constant less than or equal to c_2. Let c_2' equal c_2. Let d_1' equal d_1. Let d_2' equal d_2. Let $f(w, x, y, z) = \min\{\frac{2z}{3y}x, \frac{z}{z+y}(z-y)\}$.

Note that $B = f(c_1', c_2', d_1', d_2') = \min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$.

MC1 holds because $B \leq \frac{2d_2}{3d_1}c_2 < d_2$, since $2c_2 \leq d_1$.

MC3 holds because $B \cdot \frac{c}{c_2} \leq \frac{2d_2}{3d_1} \cdot c_2 \cdot \frac{c}{c_2} \leq c_2$.

MC4 holds because $(d_2' - B)\frac{c}{c_2} + c > (d_2' - \frac{2d_2}{3d_1}c_2)\frac{3d_1}{2d_2}\frac{c_2}{c_2} = \frac{3}{2}d_1 - c_2$. This quantity is greater than or equal to $\frac{3}{2}d_1 - \frac{1}{2}d_1$ since $d_1 \geq 2c_2$. Thus the expression is greater than or equal to d_1. MC5 holds because $(d_2' + B)\frac{c}{c_2} - c < (d_2 + \frac{2d_2}{3d_1}c_2)\frac{3d_1}{2d_2}\frac{c_2}{c_2} = \frac{3}{2}d_1 + c_2 \leq d_2$ since $\frac{3}{2}d_1 + c_2 \leq d_2$.

MC2 is not considered because c_1 is unknown. ∎

6.7. Counting with EC, MD, and CB1. Suppose methods EC, MD, and CB1 are used. The resulting asymptotic upper bound on the per-session cost is $\min\{d_2, \frac{d_2}{d_1} \cdot u + 2c_2, \frac{c_2}{c_1} \cdot u + 2c_2\}$. The only model in which exactly these methods can be used is model 11 ($\mathcal{M}[c_1, ?, d_1, d_2]$).

Corollary 6.8 below proves that the lower bound for model 11 is $(s-1) \cdot \frac{d_2}{d_2+d_1}u$. We now show this bound is asymptotically tight. As in section 6.3, we ignore the $2c_2$ term in the MD and CB1 expressions.

(Case 1) If $\frac{c_2}{c_1}u \le \frac{d_2}{d_1}u \le d_2$, the upper bound is $\Theta(\frac{c_2}{c_1}u)$. Because $\frac{d_2}{d_1}u \le d_2$, then $d_2 \le 2d_1$. Also $\frac{d_2}{d_1+d_2}u \ge \frac{d_2}{3d_1}u \ge \frac{c_2}{3c_1}u$. Thus, the lower bound is $\frac{c_2}{3c_1}u$.

(Case 2) If $\frac{c_2}{c_1}u \le d_2 < \frac{d_2}{d_1}u$, the upper bound is $\Theta(\frac{c_2}{c_1}u)$. Because $\frac{d_2}{d_1}u > d_2$, then $d_2 > 2d_1$. Also $\frac{d_2}{d_1+d_2}u \ge \frac{c_2}{c_1}u \cdot \frac{d_2-d_1}{d_2+d_1}$, which is greater than and equal to $\frac{d_2}{3c_1}u$ because $\frac{d_2-d_1}{d_2+d_1} \ge \frac{1}{3}$. Thus, the lower bound is $\frac{c_2}{3c_1}u$.

(Case 3) If $\frac{d_2}{d_1}u \le \min\{d_2, \frac{c_2}{c_1}u\}$, then the upper bound is $\Theta(\frac{d_2}{d_1}u)$. The lower bound is $\frac{d_2}{3d_1}u$ because $\frac{d_2}{d_2+d_1}u \ge \max\{\frac{d_2}{3}, \frac{d_1}{3d_2}u\}$.

(Case 4) If $d_2 < \min\{\frac{d_2}{d_1}u, \frac{c_2}{c_1}u\}$, the upper bound is $\Theta(d_2)$. The lower bound is $\frac{d_2}{3}$ because $\frac{d_2}{d_2+d_1}u \ge \max\{\frac{d_2}{3}, \frac{d_1}{3d_2}u\}$, and $\frac{d_2}{3} \le \frac{2d_2}{3d_1}c_2$.

COROLLARY 6.8. *Let c_1, d_1, and d_2 be positive reals. For $\mathcal{M}[c_1, ?, d_1, d_2]$, there exists no algorithm that solves the (s,n)-session problem within time less than $(s-1)\cdot\frac{d_2}{d_2+d_1}u$.*

Proof. Let $c = 2c_1$. Let $c_1' = c_1$. Let $c_2' = \frac{4d_2c_1}{d_2+d_1}$. Let $d_1' = d_1$. Let $d_2' = d_2$. Let $f(w,x,y,z) = \frac{z}{z+y}\cdot(z-y)$.

MC1 is satisfied because $B = f(c_1', c_2', d_1', d_2') = \frac{d_2'}{d_2'+d_1'}\cdot(d_2'-d_1') = \frac{d_2}{d_2+d_1}\cdot(d_2-d_1) < d_2$.

MC2 is satisfied because $\frac{c}{2} = c_1$.

MC4 is satisfied because $(d_2'-B)\frac{c}{c_2'} + c = (d_2 - \frac{d_2u}{d_2+d_1})\frac{2c_1}{\frac{4d_2c_1}{d_2+d_1}} + c = d_1 + c > d_1$.

MC5 is satisfied because $(d_2'+B)\frac{c}{c_2'} - c = (d_2 + \frac{d_2u}{d_2+d_1})\frac{2c_1}{\frac{4d_2c_1}{d_2+d_1}} - c = d_2 - c < d_2$.

MC3 is not considered because c_2 is not known. □

6.8. Counting with all methods. Suppose all five methods (EC, ST, MD, CB1, and CB2) are used. The resulting asymptotic upper bound on the per-session cost is $\min\{d_2, \frac{c_2}{c_1}\cdot c_2, \frac{d_2}{d_1}\cdot u + 2c_2, \frac{d_2+c_2}{d_1}\cdot c_2, \frac{c_2}{c_1}\cdot u + 2c_2\}$. The only model in which all these methods can be used is model 15 ($\mathcal{M}[c_1, c_2, d_1, d_2]$).

We now show this upper bound is asymptotically tight if

- $2c_2 \le d_1$, and
- $d_2 \le \frac{3}{2}d_1$ or $\frac{3}{2}d_1 + c_2 \le d_2$.

Corollary 6.9 shows that the per-session lower bound for model 15 is $\min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$. As in section 6.3, we ignore the $2c_2$ term in the MD and CB1 expressions.

(Case 1) If $\frac{c_2}{c_1}c_2 \le \min\{d_2, \frac{d_2}{d_1}u, \frac{d_2+c_2}{d_1}c_2, \frac{c_2}{c_1}\cdot u\}$, then the upper bound is $\Theta(\frac{c_2}{c_1}c_2)$. The lower bound is $\frac{c_2}{3c_1}c_2$ because (1) $\frac{c_2}{2c_1}c_2 \ge \frac{c_2}{3c_1}c_2$; (2) $\frac{3d_2}{2d_1}c_2 \ge \frac{d_2+c_2}{d_1}c_2 \ge \frac{c_2}{c_1}c_2$, and thus $\frac{2d_2}{3d_1}c_2 > \frac{c_2}{3c_1}c_2$; and (3) $\frac{d_2}{d_2+d_1}u \ge \max\{\frac{d_2}{3}, \frac{d_2}{3d_1}u\}$, and $\frac{d_2}{3d_1}u \ge \frac{c_2}{3c_1}c_2$.

(Case 2) If $\frac{d_2}{d_1}u \le \min\{d_2, \frac{c_2}{c_1}c_2, \frac{d_2+c_2}{d_1}c_2, \frac{c_2}{c_1}u\}$, then the upper bound is $\Theta(\frac{d_2}{d_1}u)$. The lower bound is $\frac{d_2}{3d_1}u$ because (1) $\frac{c_2}{c_1}c_2 \ge \frac{d_2}{d_1}u$, and thus $\frac{c_2}{2c_1}c_2 \ge \frac{d_2}{3d_1}u$; (2) $\frac{3d_2}{2d_1}c_2 \ge \frac{d_2+c_2}{d_1}c_2 \ge \frac{d_2}{d_1}u$, and thus $\frac{2d_2}{3d_1}c_2 > \frac{d_2}{3d_1}u$; and (3) $\frac{d_2}{d_2+d_1}u \ge \max\{\frac{d_2}{3}, \frac{d_2}{3d_1}u\}$ and $\frac{d_2}{3d_1}u$.

(Case 3) If $\frac{c_2}{c_1}u \le \min\{d_2, \frac{c_2}{c_1}c_2, \frac{d_2+c_2}{d_1}c_2, \frac{d_2}{d_1}u\}$, then the upper bound is $\Theta(\frac{c_2}{c_1}u)$. The lower bound is $\frac{4c_2}{15c_1}u$ because (1) $\frac{c_2}{2c_1}c_2 \ge \frac{c_2}{2c_1}u \ge \frac{c_2}{3c_1}u$; (2) $\frac{3d_2}{2d_1}c_2 \ge \frac{d_2+c_2}{d_1}c_2 \ge \frac{c_2}{c_1}u$, and thus $\frac{2d_2}{3d_1}c_2 > \frac{c_2}{3c_1}u$; and (3) $\frac{d_2}{d_2+d_1}u \ge \max\{\frac{d_2}{3}, \frac{d_2}{3d_1}u\}$, and $\frac{d_2}{3d_1}u \ge \frac{c_2}{3c_1}u$.

(Case 4) If $\frac{d_2+c_2}{d_1}c_2 \le \min\{d_2, \frac{c_2}{c_1}c_2, \frac{c_2}{c_1}u, \frac{d_2}{d_1}u\}$, then $\frac{d_2+c_2}{d_1}c_2 \ge \frac{d_2}{d_1}c_2$, and the upper bound is $\Theta(\frac{d_2}{d_1}c_2)$. The lower bound is $\frac{d_2}{3d_1}c_2$ because (1) $\frac{c_2}{2c_1}c_2 \ge \frac{d_2}{2d_1}c_2 \ge \frac{d_2}{3d_1}c_2$; and (2) $\frac{d_2}{d_2+d_1}u \ge \max\{\frac{d_2}{3}, \frac{d_2}{3d_1}u\}$, and $\frac{d_2}{3d_1}u \ge \frac{d_2+c_2}{3d_1}c_2 > \frac{d_2}{3d_1}c_2$; and (3) $\frac{d_2+c_2}{d_1}c_2 \ge \frac{d_2}{d_1}c_2 \ge \frac{d_2}{3d_1}c_2$.

(Case 5) If $d_2 \geq \min\{\frac{c_2}{c_1}c_2, \frac{c_2}{c_1}u, \frac{d_2+c_2}{d_1}c_2, \frac{d_2}{d_1}u\}$, then the upper bound is $\Theta(d_2)$. The lower bound is $\frac{d_2}{3}$ because (1) $\frac{c_2}{2c_1}c_2 \geq \frac{d_2}{3}$; (2) $\frac{d_2}{d_2+d_1}u \geq \max\{\frac{d_2}{3}, \frac{d_2}{3d_1}u\}$; and (3) $\frac{3d_2}{2d_1}c_2 \geq \frac{d_2+c_2}{d_1}c_2 \geq d_2$, and $\frac{3d_2}{3d_1}c_2 > \frac{d_2}{3}$.

COROLLARY 6.9. *Let c_1, c_2, d_1, and d_2 be positive reals such that $2c_2 \leq d_1$ and either $d_2 \leq \frac{3}{2}d_1$ or $\frac{3}{2}d_1 + c_2 \leq d_2$. For $\mathcal{M}[c_1, c_2, d_1, d_2]$, there exists no algorithm that solves the (s, n)-session problem within time less than* $\min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}(s-1)$.

Proof.

Case 1. $c_2 \geq \min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$.

The lower bound clearly holds because all processes have to take at least s steps to solve the (s, n)-session problem and c_2 is the upper bound on step time.

Case 2. $c_2 < \min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$.

Note that in this case, by the hypothesis of the corollary, the following are true:

$$d_1 \geq 2c_2, \tag{6.4}$$

$$d_2 \geq \frac{3d_1}{2} + c_2. \tag{6.5}$$

Let $c = \max\{2c_1, \frac{3d_1}{2d_2}c_2\}$. Let $c_1' = c_1$. Let $c_2' = c_2$. Let $d_1' = d_1$. Let $d_2' = d_2$. Let $f(w, x, y, z) = \min\{\frac{x}{2w}x, \frac{2z}{3y}x, \frac{z}{z+y}(z-y)\}$.

We show that they satisfy the hypothesis of Theorem 3.1.

Note that $B = f(c_1', c_2', d_1', d_2') = \min\{\frac{c_2}{2c_1}c_2, \frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$.

Case 2.1. Suppose $\frac{c_2}{2c_1}c_2 < \min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$. Then $B = \frac{c_2}{2c_1}c_2$ and $c = 2c_1$. Furthermore,

$$\frac{c_2}{2c_1} \leq \frac{2d_2}{3d_1}. \tag{6.6}$$

For MC1, clearly $B = \frac{c_2}{2c_1}c_2 \leq \frac{d_2}{d_2+d_1}u < d_2$.

For MC2, $\frac{c}{2} = c_1$.

For MC3, $B\frac{c}{c_2'} = c_2$.

For MC4,

$$\begin{aligned}
(d_2' - B)\tfrac{c}{c_2'} + c &= d_2\tfrac{2c_1}{c_2} - c_2 + 2c_1 \\
&\geq d_2\tfrac{3d_1}{2d_2} - c_2 && \text{because of (6.6)} \\
&\geq \tfrac{3d_1}{2} - \tfrac{d_1}{2} && \text{because of (6.4)} \\
&= d_1.
\end{aligned}$$

For MC5, it suffices to prove that $d_2 - ((d_2' + B)\frac{c}{c_2'} - c) \geq 0$:

$$\begin{aligned}
d_2 - ((d_2' + B)\tfrac{c}{c_2'} - c) &\geq d_2 - ((d_2 + \tfrac{c_2^2}{2c_1})\tfrac{2c_1}{c_2} - 2c_1) \\
&= d_2(1 - \tfrac{2c_1}{c_2}) - (c_2 - 2c_1) \\
&\geq 0 && \text{because of (6.4).}
\end{aligned}$$

Case 2.2. Suppose $\frac{c_2}{2c_1}c_2 \geq \min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$. Then $B = \min\{\frac{2d_2}{3d_1}c_2, \frac{d_2}{d_2+d_1}u\}$ and $c = \frac{3d_1}{2d_2}c_2$. Furthermore,

$$\frac{c_2}{2c_1} > \frac{2d_2}{3d_1}. \tag{6.7}$$

For MC1, $B \leq \frac{d_2}{d_2+d_1}u < d_2$.
For MC2, $\frac{c}{2} = \frac{1}{2}\frac{3d_1}{2d_2}c_2 \geq c_1$ because of (6.7).
For MC3,

$$B\frac{c}{c_2'} \leq \frac{2d_2}{3d_1}c_2\frac{\frac{3d_1}{2d_2}c_2}{c_2} = c_2.$$

For MC4,

$$\begin{aligned}(d_2' - B)\tfrac{c}{c_2'} + c &\geq (d_2' - \tfrac{2d_2}{3d_1}c_2)\tfrac{c}{c_2'} + c \\ &= \tfrac{3d_1}{2} - c_2 + c \\ &> \tfrac{3d_1}{2} - c_2 \\ &\geq \tfrac{3d_1}{2} - \tfrac{d_1}{2} \qquad \text{because of (6.4)} \\ &= d_1.\end{aligned}$$

For MC5,

$$\begin{aligned}(d_2' + B)\tfrac{c}{c_2'} - c &\leq (d_2' + \tfrac{2d_2}{3d_1}c_2)\tfrac{c}{c_2'} - c \\ &= \tfrac{3d_1}{2} + c_2 - c \\ &< \tfrac{3d_1}{2} + c_2 \\ &\leq d_2 \qquad \text{because of (6.5).} \quad \square\end{aligned}$$

7. Conclusion and discussion. This paper concerns timing models in distributed systems that lie between the synchronous and asynchronous models. Four timing parameters are considered: the maximum and minimum process step times and message delays. Timing models are obtained by considering independently whether each parameter is known (i.e., is hard-wired into the processes' code) or unknown, giving rise to four SM models and 16 MP models.

The session problem is an abstraction of synchronization problems in distributed systems. It has been used as a test case to demonstrate the differences in the time needed to solve problems in various timing models, for both SM and MP systems. In this paper, we continue to use the session problem to compare quantitatively a family of models in which various parameters are either known or unknown.

For each unknown parameter model, we obtain an asymptotically tight time complexity bound on the session problem. Two of the algorithms were previously known, while the other three are new. We categorize the algorithms in terms of "ways to count." The intuition is that processes must have some way to count the passage of other processes' steps in order to "know" when a session has occurred. Our matching lower and upper bounds indicate that the algorithms are the optimal ways to count and allow us to construct a lattice of timing models in terms of the counting algorithms that are applicable to a model (cf. Figure 1.1). This hierarchy confirms the common belief that as a model has more timing knowledge, it behaves more like the synchronous model.

All but one of our lower bound results are new and all of them are obtained by one modular lower bound proof. The lower bound technique combines those of [3] for the asynchronous model in the SM system and [5] for the asynchronous and semisynchronous models in the MP systems. Our technique identifies one sufficient condition for a lower bound in a timing model to hold in solving the (s, n)-session problem and is applicable to every unknown parameter model as well as to those models previously studied for the session problem.

For several unknown parameter models, we were not able to show the lower bound without making some assumptions about the timing parameters. It will be interesting to develop a new lower bound technique that can show either the same or tighter lower bounds without such assumptions.

It will be also interesting to see whether our modular lower bound proof technique can be applied to show lower bounds for other distributed computing problems, such as the mutual exclusion problem and the dining philosophers problem, in many different timing models.

Appendix A. Communication in SM. Consider an $(a-1)$-ary tree with n leaves in which each level, except possibly the lowest, has the maximum number of nodes. The number of levels in the tree is $\lceil \log_{a-1} n \rceil + 1$. Associated with each node in the tree are a process and a shared variable. Each port process and its port variable are associated with a leaf node; the processes and variables associated with internal nodes are called *relay* processes and variables. The relay variable associated with a node is accessed by the process associated with the node and the processes associated with that node's children in the tree. Figure A.1 illustrates a tree with $a = 4$ and $n = 7$.

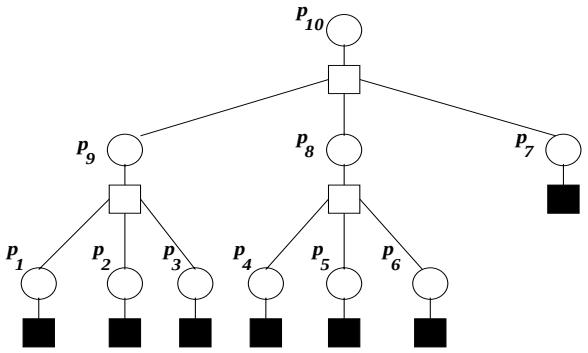


FIG. A.1. *A tree network with $a = 4$ and $n = 7$, where circles represent processes, empty squares relay variables, dark squares port variables, and solid lines memory access patterns of processes. Port processes are p_1 through p_7.*

Each relay variable has two fields, *up* and *down*. Each process has two local variables, *lup* and *ldown*; initially they are empty except that *lup* at a port process holds the information to be propagated.

Each relay process p other than the root repeats the following two steps. First, p accesses its own shared variable, saving the contents of the *up* field in *lup* and appending the contents of *ldown* to the *down* field. Second, p accesses its parent's variable, appending *lup* to the *up* field and saving the *down* field in *ldown*.

The root continuously accesses its own variable; at each access it copies the *up* field to the *down* field.

In the example of Figure A.1, a piece of information m is transferred from p_2 to p_6 as follows: p_2 appends m to $v_9.up$; p_9 obtains the contents of $v_9.up$ and appends them to $v_{10}.up$; p_{10} copies the contents of $v_{10}.up$ to $v_{10}.down$; p_8 obtains the contents of $v_{10}.down$ and appends them to $v_8.down$; p_6 obtains the contents of $v_8.down$ and gets m.

It takes at most $c_2 \cdot \lceil \log_{a-1} n \rceil$ time for a piece of information (or a "message") to be relayed up to the root for the following reason. In each time interval of length c_2,

every process takes at least one step, and thus every relay process other than the root passes the message up to its parent. Likewise, it takes additional $c_2 \cdot (\lceil \log_{a-1} n \rceil + 1)$ time for the message at the root to be relayed down to a leaf node in the tree (the one additional c_2 is for the root to move the message from its *up* to its *down*). Thus, the broadcast is accomplished in $c_2 \cdot \Theta(\log_{a-1} n)$ time. A similar tree network is mentioned in [3].

When we say *broadcast* in the SM model, it implies all of the interactions between processes in the tree network that are needed to accomplish the broadcasting. We only describe the role of port processes in an algorithm and assume that broadcast encapsulates the interactions among port processes and other processes which participate in the tree-network communication. In addition, we use the term "step" interchangeably with "port step"; when necessary, we make the proper distinction.

Appendix B. Correctness proofs of counting methods.

B.1. Correctness proof of CB1.

THEOREM B.1. *CB1 solves the (s, n)-session problem within time $(s-2) \cdot (\frac{c_2}{c_1} u + u + 2c_2) + d_2 + 3c_2$ if c_1, d_1, and d_2 are known.*

Proof. Consider an arbitrary admissible timed computation C of technique CB1.

For each k, $0 \le k \le s-1$, let p_{i_k} be the first process that sets its *session* variable to k in C. To increment *session*, a process must receive a set of messages that satisfy condition 1 in Figure 4.3. Let M_k be the set of messages received by p_{i_k} that cause p_{i_k} to set $session_{i_k}$ to k (M_0 is the empty set); let m_k be the message which is sent last among M_k (if there is a tie, choose an arbitrary message among them).

LEMMA B.2. *Let π be the step which sends m_k. There are at least k sessions by the time that π occurs in C.*

Proof. We proceed by induction on k. For the basis, when $k = 0$, it is always true that there are at least 0 sessions in C.

Inductively when $k > 0$, assuming the lemma is true for $k-1$, we show that when π occurs, there are at least k sessions.

Let τ be the step that sends m_{k-1} and let σ be the step in which $p_{i_{k-1}}$ sets $session_{i_{k-1}}$ to $k-1$. Such a step exists because $session_{i_{k-1}}$ is always incremented by 1. For $p_{i_{k-1}}$ to update $session_{i_{k-1}}$, condition 1 in Figure 4.3 must hold.

Let t be the time when τ occurs and t' be the time when σ occurs.

The message m_{k-1} must arrive at $buf_{i_{k-1}}$ at time between $[t+d_1, t+d_2]$ because of the bounds on message delays. Thus, $t' - t \ge d_1$ because σ occurs after $p_{i_{k-1}}$ receives m_{k-1}. Note that *count* in the algorithm is reset whenever *session* is updated. Let t'' be the time that p_{i_k} sets $session_{i_k}$ to k. From the code, because condition 1 should be true before *session* is updated, $count_{i_k}$ must equal to B at time t''. Thus, when $count_{i_k}$ is equal to B, at least $B \cdot c_1$ time has elapsed since time t' because t' is the time that *session* is updated to $k-1$ for the first time in computation C and $count_{i_k}$ must be reinitialized after time t. Thus, $t'' \ge t' + Bc_1 = t' + d_2 - d_1 \ge t + d_2$ because $t' - t \ge d_1$.

Therefore, the difference between times t and t'' is bigger than d_2. Thus, all messages received at time t'' or later must be sent after time t, at which time there were $k-1$ sessions by the inductive assumption. Since at least one message is sent by each process after time t, there must be at least one additional step by all processes between time t and the time π occurs. Therefore, there are at least k sessions by the time π occurs. □

From Lemma B.2, it follows that there are at least $s-1$ sessions at the time that m_{s-1} is sent. All processes will eventually set their *session*'s to $s-1$. Since

all processes take additional steps after there are at least $s-1$ sessions (to receive a message), there are at least s sessions in C. Thus the algorithm is correct.

We now calculate the running time of the algorithm. We define for each k, $2 \le k \le s-1$, $T_k = \max\{t : p_i \text{ sets } session_i \text{ to } k \text{ at time } t \text{ in } C \text{ for all } p_i \in R\}$. T_k is the latest time that a process sets *session* to k.

LEMMA B.3. *For each k, $2 \le k \le s-1$, $T_{k+1} \le T_k + \frac{c_2}{c_1}u + u + 2c_2$.*

Proof. After a process p_i sets $session_i$ to k, it takes at most $\frac{u}{c_1}c_2$ time for *count* to be bigger than B. Then it takes at most Δ time additionally for condition 1 to be true (i.e., for at least one message from every process to be received after *count* becomes bigger than B). We prove that $\Delta \le u + 2c_2$.

Let m_1 be the message that is received from process p_j by p_i just before condition 1 becomes true in p_i (i.e., p_i has waited $\frac{u}{c_1}c_2$ time). Message m_1 exists because condition 1 becomes true only if there are enough messages in *msgs*, which is emptied only after condition 1 becomes true. Let t be the time that m_1 is sent. p_j must broadcast another message m_2 within $t + c_2$ to process i because according to the code, all processes broadcast a message at every step. m_2 will be delivered to buf_i by time $t + c_2 + d_2$ (because it takes at most d_2 delay for a message to arrive at a buffer) and be received by time $t + 2c_2 + d_2$ (because it takes at most c_2 time for a process to take a step).

Process p_i will receive m_1 at time bigger than or equal to $t + d_1$ because it is sent at time t and it takes at least d_1 time delay for a message to arrive at a buffer. Since process i receives m_2 by time $t + 2c_2 + d_2$, the maximum time difference between the time that process i receives m_1 and the time that it receives m_2 is $(t + 2c_2 + d_2) - (t + d_1) = d_2 - d_1 + 2c_1 = u + 2c_2$. Therefore, $\Delta = u + 2c_2$. □

By the algorithm, initially it takes at most $d_2 + 2c_2$ time to receive at least one message from all processes in order to accomplish the first session. Therefore $T_1 = d_2 + 2c_1$. Using Lemma B.3, T_{s-1}, which is the latest time that a process sets *session* to $s-1$, is at most $(s-2)\cdot(\frac{c_2}{c_1}u + u + 2c_2) + T_1$. After T_{s-1}, a process takes one step to complete s sessions. Therefore, a process enters the idle state by time $(s-2)\cdot(\frac{c_2}{c_1}u + u + 2c_2) + d_2 + 2c_2 + c_2$. □

B.2. Correctness proof of MD.

THEOREM B.4. *MD solves the (s,n)-session problem within time $(s-2)\cdot(\frac{d_2}{d_1}u + u + 2c_2) + d_2 + 2c_2$ if d_1 and d_2 are known.*

Proof. MD differs from CB1 only in the way that *count* is incremented and in that B is set to u/d_1. The rest of the code is the same. The correctness proof is similar to that of technique CB1.

Since *count* and B affect condition ($count \ge B$) in the code, we only need to prove that when $B < count$, at least time u has passed since the last time *session* was incremented. *count* is incremented to k only when a process receives $m(j, session, k-1)$ for any j and for some value of *session*. When we apply this argument inductively, we prove that there must be a chain of processes $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$, where p_{i_r} receives $m(i_{r-1}, session, r-1)$ from $p_{i_{r-1}}$. Thus, when *count* is B, at least time $Bd_1 = d_2 - d_1 = u$ has passed after p_{i_1} sent $m(i_1, session, 1)$ because it takes at least d_1 message delay for a message to be received after it is sent.

For time complexity, it takes at most time $d_2 + c_2$ for p_{i_k} to receive $m(i_{k-1}, session, k-1)$ after $p_{i_{k-1}}$ receives $m(i_{k-2}, session, k-2)$. Therefore, for *count* to be bigger than or equal to B, it takes at most $B(d_2 + c_2)$. After that, for condition 1 in the code to be true it takes at most $(u + 2c_2)$, as proved in the proof of Lemma B.3. Therefore, the running time of technique MD is $(s-2)\cdot(\frac{d_2+c_2}{d_1}u + u + 2c_2) + d_2 + 2c_2$. □

B.3. Correctness proof of CB2.

THEOREM B.5. *Technique CB2 solves the* (s,n)*-session problem in time* $(s-1)\cdot \frac{d_2+c_2}{d_1}c_2 + c_2$ *if* c_2 *and* d_1 *are known.*

Proof. count is incremented to k only when a process receives $m(j,*,k-1)$ for any j. When we apply this argument inductively, we prove that there must be a chain of processes $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$, where p_{i_r} receives $m(i_{r-1},*,r-1)$ from $p_{i_{r-1}}$. Thus, when *count* is B, at least time $Bd_1 = \frac{c_2}{d_1}(s-1)d_1$ has passed after p_{i_1} sent $m(i_1,*,1)$ because it takes at least d_1 message delay for a message to be received after it is sent. Thus, when $count > B$, at least time $c_2(s-1)$ has passed since the start of the computation. The theorem follows. □

REFERENCES

[1] R. ALUR, H. ATTIYA, AND G. TAUBENFELD, *Time-adaptive algorithms for synchronization*, SIAM J. Comput., 26 (1997), pp. 539–556.

[2] H. ATTIYA, C. DWORK, N. A. LYNCH, AND L. J. STOCKMEYER, *Bounds on the time to reach agreement in the presence of timing uncertainty*, J. ACM, 41 (1994), pp. 122–152.

[3] E. ARJOMANDI, M. FISCHER, AND N. A. LYNCH, *Efficiency of synchronous versus asynchronous distributed systems*, J. ACM, 30 (1983), pp. 449–456.

[4] H. ATTIYA AND N. A. LYNCH, *Time bounds for real-time process control in the presence of timing uncertainty*, Inform. and Comput., 110 (1994), pp. 183–232.

[5] H. ATTIYA AND M. MAVRONICOLAS, *Efficiency of semi-synchronous versus asynchronous networks*, Math. Systems Theory, 27 (1994), pp. 547–571.

[6] R. ALUR AND G. TAUBENFELD, *Fast timing-based algorithms*, Distrib. Comput., 10 (1996), pp. 1–10.

[7] G. BAUDET, *Asynchronous interactive methods for multi-processors*, J. ACM, 32 (1978), pp. 226–244.

[8] B. A. COAN AND G. THOMAS, *Agreeing on a leader in real-time*, in Proceedings of the 11th Real-Time Systems Symposium, Lake Buena Vista, FL, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 1–7.

[9] B. A. COAN AND J. L. WELCH, *Transaction commit in a realistic timing model*, Distrib. Comput., 4 (1990), pp. 87–103.

[10] D. DOLEV, C. DWORK, AND L. STOCKMEYER, *On the minimal synchronism needed for distributed consensus*, J. ACM, 34 (1987), pp. 77–97.

[11] C. DWORK, N. LYNCH, AND L. STOCKMEYER, *Consensus in the presence of partial synchrony*, J. ACM, 35 (1988), pp. 288–323.

[12] M. FISCHER, N. LYNCH, AND M. PATERSON, *Impossibility of distributed consensus with one faulty process*, J. ACM, 32 (1985), pp. 374–382.

[13] J. GOODMAN, M. VERNON, AND P. WOEST, *Efficient synchronization primitives for large-scale cache coherent multiprocessors*, in Proceedings of the 3rd International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, MA, 1989, pp. 64–75.

[14] N. LYNCH, *Distributed Algorithms*, Morgan-Kaufmann, San Francisco, CA, 1996.

[15] N. LYNCH AND N. SHAVIT, *Timing-based mutual exclusion*, in Proceedings of the 13th Real-Time Systems Symposium, San Antonio, TX, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 2–11.

[16] M. MAVRONICOLAS, *Efficiency of semi-synchronous versus asynchronous systems: Atomic shared memory*, Comput. Math. Appl., 25 (1993), pp. 81–91.

[17] S. PONZIO, *Consensus in the presence of timing uncertainty: Omission and Byzantine failures*, in Proceedings of the ACM Symposium on Principles of Distributed Computing, Montreal, QC, Canada, 1991, pp. 125–137.

[18] I. RHEE AND J. L. WELCH, *The impact of time on the session problem*, in Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing, Vancouver, BC, Canada, Association for Computing Machinery, New York, 1992, pp. 191–201.

[19] I. RHEE AND J. L. WELCH, *Time bounds on synchronization in a periodic distributed system*, Inform. Process. Lett., 64 (1997), pp. 87–93.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 1040–1049

MULTIRATE REARRANGEABLE CLOS NETWORKS AND A GENERALIZED EDGE-COLORING PROBLEM ON BIPARTITE GRAPHS*

HUNG Q. NGO† AND VAN H. VU‡

Abstract. Chung and Ross [*SIAM J. Comput.*, 20 (1991), pp. 726–736] conjectured that the minimum number $m(n,r)$ of middle-stage switches for the symmetric 3-stage Clos network $C(n, m(n,r), r)$ to be rearrangeable in the multirate environment is at most $2n-1$. This problem is equivalent to a generalized version of the bipartite graph edge-coloring problem. The best bounds known so far on this function $m(n,r)$ are $11n/9 \le m(n,r) \le 41n/16 + O(1)$, for $n, r \ge 2$, derived by Du et al. [*SIAM J. Comput.*, 28 (1999), pp. 464–471]. In this paper, we make several contributions. First, we give evidence to show that even a stronger result might hold. In particular, we give a coloring algorithm to show that $m(n,r) \le \lceil (r+1)n/2 \rceil$, which implies $m(n,2) \le \lceil 3n/2 \rceil$—stronger than the conjectured value of $2n-1$. Second, we derive that $m(2,r) = 3$ by an elegant argument. Last, we improve both the best upper and lower bounds given above: $\lceil 5n/4 \rceil \le m(n,r) \le 2n-1+\lceil (r-1)/2 \rceil$, where the upper bound is an improvement over $41n/16$ when r is relatively small compared to n. We also conjecture that $m(n,r) \le \lfloor 2n(1-1/2^r) \rfloor$.

Key words. Clos network, multirate rearrangeable, bipartite graph, generalized edge-coloring

AMS subject classifications. 05C15, 05C85, 68R10, 94C15

DOI. 10.1137/S0097539702408235

1. Introduction.

The Clos network has been widely used for data communications and parallel computing systems. Quite a lot of research efforts [1, 2, 3, 5, 6, 9, 10, 11, 13, 14, 15, 16, 17, 22] have been put into investigating the nonblocking properties and rearrangeability of the Clos network. The 3-stage Clos network was paid special attention to since it can be expanded in a "straightforward" way to the multistage Clos network. Recently, Ngo and Pan [18] observed that the 3-stage Clos network is "equivalent" to the wavelength division multiplexed (WDM) split cross-connects [20, 21], giving new applications to the classic Clos networks. Let us first formally introduce some related concepts.

The Clos network $C(n_1, r_1, m, n_2, r_2)$ is a 3-stage interconnection network, where the first stage consists of r_1 crossbars of size $n_1 \times m$, the last stage has r_2 crossbars of dimension $m \times n_2$, and the middle stage has m crossbars of dimension $r_1 \times r_2$ (see Figure 1). Each input switch I_i $(i = 1, \ldots, r_1)$ is connected to each middle switch M_j $(j = 1, \ldots, m)$. Similarly, the middle stage and the last stage are fully connected. When $n_1 = n_2 = n$ and $r_1 = r_2 = r$, the network is called the *symmetric 3-stage Clos network*, denoted by $C(n, m, r)$. Any switch is assumed to be nonblocking; i.e., any inlet can be connected to any outlet as long as there is no conflict on the outlet. A switch of dimension $p \times q$ could be thought of as a crossbar of size $p \times q$ with pq cross-points. Having too many cross-points is expensive, and we would like to design a huge switch using smaller switches with fewer cross-points than when a brute-force

*Received by the editors May 23, 2002; accepted for publication (in revised form) April 29, 2003; published electronically July 8, 2003. A previous version of this paper appeared in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Baltimore, MD, 2003, ACM, New York, 2003, pp. 834–840.
http://www.siam.org/journals/sicomp/32-4/40823.html

†Computer Science and Engineering Department, 201 Bell Hall State University of New York at Buffalo, Amherst, NY (hungngo@cse.buffalo.edu).

‡Department of Mathematics, University of California at San Diego, 9500 Gilman Dr., La Jolla, CA 92093-0112 (vanvu@math.ucsd.edu).

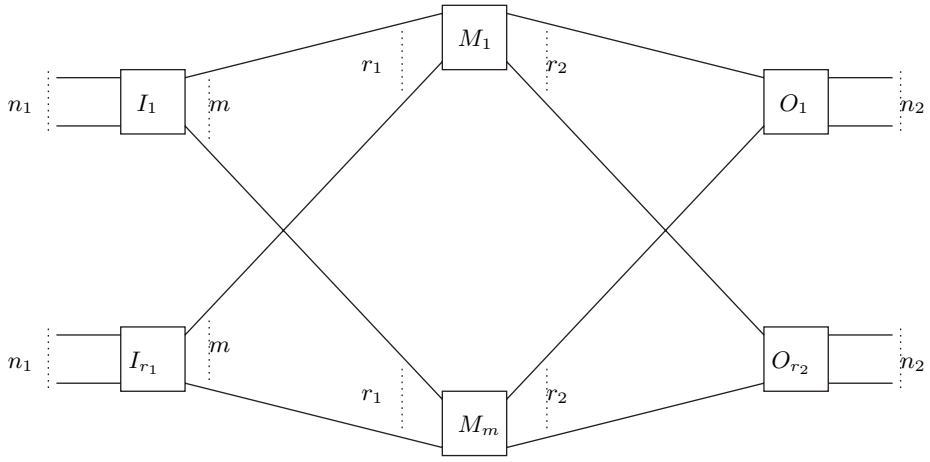


FIG. 1. *The 3-stage Clos network* $C(n_1, r_1, m, n_2, r_2)$.

design is used. The inlets (outlets) of the input (output) switches are the *inputs* (*outputs*) of the network. Inputs and outputs are referred to as *external links*, while links between switches are referred to as *internal links*.

In the multirate environment, a *connection request* is a triple (i, j, w), where i is an inlet, j an outlet, and w the weight. A *request frame* is a collection of requests such that the total weight of all requests in the frame involving a fixed inlet or outlet does not exceed unity. To discuss routing it is convenient to assume that all links are directed from left to right. Thus a *path* from an inlet to any outlet always consists of the following sequence: an inlet link $\rightarrow$ an input switch $\rightarrow$ a link $\rightarrow$ a center switch $\rightarrow$ a link $\rightarrow$ an output switch $\rightarrow$ an outlet link. Furthermore, since the crossbars are assumed to be nonblocking, a request (i, j, w) is *routable* if and only if there exists a path from i to j such that every link on this path has unused capacity at least $1 - w$ before carrying out this request. A request frame is routable if there exists a set of paths, one for each request, such that for every link the sum of weights of all requests going through it does not exceed unity. The Clos network $C(n, m, r)$ is said to be *multirate rearrangeable* (or just rearrangeable, as in this paper we consider only the multirate environment) if *every* request frame is routable.

Let $m(n, r)$ denote the minimum value of m such that $C(n, m, r)$ is multirate rearrangeable for $n, r \geq 2$. (The cases where either n or r are 1 are trivial; hence we consider only $n, r \geq 2$ from here on.) Our problem is to find $m(n, r)$ or at least some good bounds for this function.

The problem appears to be difficult. Let us preview some previous works on this problem. Melen and Turner [16] initiated the research on multirate switching networks. In 1991, Chung and Ross [3] conjectured that $m(n, r) \leq 2n - 1$ and until now no one has been able to prove or disprove the conjecture. The best bounds known so far on this function $m(n, r)$ were obtained by Du et al. [5]:

$$11n/9 \leq m(n, r) \leq 41n/16 + O(1).$$

Lin et al. [14] confirmed Chung–Ross conjecture for a restricted discrete bandwidth case where each connection has a weight chosen from a set $\{1 \geq w_1 > \cdots > w_h > 1/2 \geq w_{h+1} > \cdots > w_k\}$ which satisfies the condition that w_i is an integer multiple of w_{i+1} for $i = h + 1, \ldots, k - 1$. Hu et al. [10] studied the monotone routing strategy

and showed that under this strategy

$$m(n,r) \le 2n+1 \quad \text{for } n=2,3,4, \tag{1}$$

$$m(n,r) \le 2n+3 \quad \text{for } n=5,6. \tag{2}$$

Ngo [17] proposed the grouping algorithm which shows that $m(n,r) \le 2n-1+r$ and that $m(n,r) \le 2n + \frac{n-1}{2^k}$ whenever $r \le \frac{n}{2^k-1}$.

In this paper, we give evidence to show that a stronger version of Chung-Ross conjecture might hold. In particular, we show that $m(n,r) \le \lceil \frac{(r+1)n}{2} \rceil$, which implies $m(n,2) \le \lceil \frac{3n}{2} \rceil$. This is stronger than the conjectured value of $2n-1$. We conjecture that

$$m(n,r) \le \left\lfloor 2n\left(1-\frac{1}{2^r}\right)\right\rfloor, \quad n,r \ge 2.$$

We believe that the new conjectured upper bound is also the correct value for $m(n,r)$. Second, we verify that Chung and Ross were right on target when $n=2$, i.e., $m(2,r)=3$, by a new elegant argument. Last, we give better upper and lower bounds for the general case:

$$\left\lceil \frac{5n}{4} \right\rceil \le m(n,r) \le 2n-1+\left\lceil \frac{r-1}{2} \right\rceil.$$

All these are done in the context of a generalized version of the edge-coloring problem on weighted bipartite graphs to be introduced in the next section. These weighted graphs have maximum degree n in the weighted sense.

As a side note, Ngo and Pan [18] showed that the 3-stage Clos network is equivalent to the WDM split cross-connects [20, 21] under this multirate environment; hence the results in this paper also apply to the split cross-connects. Each rate can be thought of as the bandwidth fraction of a wavelength obtained from time division multiplexing.

2. A generalized bipartite graph edge-coloring problem. Given a request frame $\mathcal{F}$, define a weighted bipartite multigraph $G_{\mathcal{F}} = (I, O; E)$, where I (respectively, O) contains all the input (respectively, output) switches. There is an edge with weight w between vertices X, Y of G for each request (x, y, w), where x (respectively, y) is an inlet (respectively, outlet) of X (respectively, Y). $C(n,m,r)$ is rearrangeable if and only if for all $\mathcal{F}$ the edges of $G_{\mathcal{F}}$ can be m-colored such that at every vertex the total weight of edges of the same color incident to this vertex is at most unity. To see this, just associate each color with a center switch.

We now formally define the equivalent bipartite graph edge-coloring problem. Throughout this paper we assume $n, r \ge 2$ are integers. Let $\mathcal{B}_r^n$ be the collection of edge-weighted $r \times r$ bipartite multigraphs $G = (A, B; E)$ ($|A| = |B| = r$) with weight function $w : E \to (0,1]$ satisfying the condition that for every $v \in V(G) = A \cup B$, the set $I(v)$ of edges incident to v can be partitioned into n groups $g(v,i)$, $1 \le i \le n$, such that

$$\sum_{e \in g(v,i)} w(e) \le 1 \quad \forall i = 1, \ldots, n. \tag{3}$$

We shall refer to condition (3) as the *grouping condition*. The grouping condition simply refers to the fact that the total weight of all requests from an inlet or to an outlet is at most unity.

A k-edge-coloring of $G \in \mathcal{B}_r^n$ is a coloring $l : E(G) \to C$, where C is a set of k colors, such that for every $v \in V(G)$ and every color $c \in C$

$$\sum_{\substack{e \in I(v) \\ l(e)=c}} w(e) \leq 1. \tag{4}$$

Let $m(n,r)$ be the minimum integer k such that every $G \in \mathcal{B}_r^n$ is k-edge-colorable. Our job is to find good bounds for $m(n,r)$ or the exact value if possible. Notice that when all the weights are 1, this problem reduces to the edge-coloring of a bipartite graph with maximum degree at most n. Thus, $m(n,r) = n$ when the weights are all unity. This can be shown as a trivial consequence of P. Hall's matching condition or of König's line coloring theorem [12].

3. A new lower bound. The main result of this section is the following theorem.

THEOREM 3.1. *For integers $n, r \geq 2$, we have*

$$m(n,r) \geq m(n,2) \geq \left\lceil \frac{5n}{4} \right\rceil \text{ when } n \text{ is even}$$

and

$$m(n,r) \geq m(n,2) \geq \left\lceil \frac{5n-1}{4} \right\rceil \text{ when } n \text{ is odd.}$$

Proof. The natural approach to find a lower bound k for $m(n,r)$ is to find a particular graph $G \in \mathcal{B}_r^n$ which requires at least k colors. The fact that $m(n,r) \geq m(n,2)$ is trivial. To show the inequality for even n, consider the following graph $G \in \mathcal{B}_r^2$:

- $G = (\{1,2\}, \{1',2'\}; E)$.
- There are n edges from 1 to $1'$ with weight 0.6.
- There are n edges from 1 to $2'$ with weight 0.4.
- There are $n/2$ edges from 2 to $2'$ with weight 1.

The grouping condition is easily seen to be satisfiable. The 0.6-edges in $I(1)$ require n colors. Let k be the number of colors shared by the 0.6-edges and 0.4-edges of $I(1)$. Then, looking from vertex 1 we need at least $n + \frac{n-k}{2}$ colors. On the other hand, looking from vertex $2'$ we need at least $\frac{n}{2} + k + \frac{n-k}{2}$ colors. Consequently, the total number of colors needed is at least

$$\max\left\{ n + \frac{n-k}{2}, \quad \frac{n}{2} + k + \frac{n-k}{2} \right\} \geq \frac{(n + \frac{n-k}{2}) + (\frac{n}{2} + k + \frac{n-k}{2})}{2}$$
$$= \frac{5n}{4}.$$

The case when n is odd can be shown similarly. ∎

4. The exact value of $m(2,r)$. The main result of this section is an algorithm to color all graphs in $\mathcal{B}_r^2$ using at most three colors.

THEOREM 4.1. *When $r \geq 2$, we have*

$$m(2,r) = 3.$$

Proof. Theorem 3.1 implies $m(2,r) \geq 3$. We are left to show that every graph $G \in \mathcal{B}_r^2$ is 3-colorable. For $G = (A, B; E) \in \mathcal{B}_r^2$, let $A = B = \{1, 2, \ldots, r\}$. The grouping condition indicates that edges incident to each vertex v could be partitioned into two groups $g(v,1)$ and $g(v,2)$ with total weight at each group at most 1. For $i, j \in \{1,2\}$ and $a \in A, b \in B$, let

$$w_{ij}(a,b) = \sum_{\substack{e=(a,b)\in E \\ e\in g(a,i)\cap g(b,j)}} w(e). \tag{5}$$

In other words, $w_{ij}(a,b)$ is the total weight of all edges e from $a \in A$ to $b \in B$, where e belongs to group i of vertex a and group j of vertex b. The grouping condition implies that for a fixed $i_0 \in \{1,2\}$ and $a_0 \in A$, we have

$$\sum_{b\in B} (w_{i_0 1}(a_0, b) + w_{i_0 2}(a_0, b)) \leq 1. \tag{6}$$

Similarly, for a fixed $j_0 \in \{1,2\}$ and $b_0 \in B$, we get

$$\sum_{a\in A} (w_{1 j_0}(a, b_0) + w_{2 j_0}(a, b_0)) \leq 1. \tag{7}$$

Clearly, the number of colors needed to color G does not change if at any vertex $v \in V$ we relabel the groups $g(v,1)$ and $g(v,2)$. (Namely, group 1 becomes group 2 and vice versa.) This relabelling does change the values $w_{ij}(v,b)$ or $w_{ij}(a,v)$, though. Now, relabel the groups at all vertices of G to maximize the following sum:

$$\sum_{\substack{a\in A, \\ b\in B}} (w_{11}(a,b) + w_{22}(a,b)). \tag{8}$$

To this end, we use three colors to color all edges of G as follows:

- One color is for all edges in

$$\bigcup_{\substack{a\in A, \\ b\in B}} (g(a,1) \cap g(b,1)). \tag{9}$$

- Another color is for all edges in

$$\bigcup_{\substack{a\in A, \\ b\in B}} (g(a,2) \cap g(b,2)). \tag{10}$$

- The last color is for all edges in

$$\bigcup_{\substack{a\in A, \\ b\in B}} (g(a,1) \cap g(b,2)) \bigcup \bigcup_{\substack{a\in A, \\ b\in B}} (g(a,2) \cap g(b,1)). \tag{11}$$

It is straightforward to verify that all edges belong to one of the three color classes. To show that this is a valid coloring, we shall verify that the total weight of edges at each color class which are incident to the same vertex is at most 1. The total weight of edges of color class (9) which are incident to vertex $a \in A$ is

$$\sum_{b\in B} w_{11}(a,b) \leq \sum_{b\in B} (w_{11}(a,b) + w_{12}(a,b)) \leq 1.$$

The cases of color class (9) with a vertex $b \in B$ and of color class (10) are done similarly.

Last, the total weight of edges of color class (11) which are incident to vertex $a \in A$ is

$$\sum_{b \in B} \left(w_{12}(a,b) + w_{21}(a,b) \right). \tag{12}$$

If this sum is > 1, then

$$\sum_{b \in B} \left(w_{11}(a,b) + w_{22}(a,b) \right) < 1, \tag{13}$$

since

$$\begin{aligned} &\sum_{b \in B} \left(w_{12}(a,b) + w_{21}(a,b) \right) + \sum_{b \in B} \left(w_{11}(a,b) + w_{22}(a,b) \right) \\ &= \sum_{b \in B} \left(w_{11}(a,b) + w_{12}(a,b) \right) + \sum_{b \in B} \left(w_{21}(a,b) + w_{22}(a,b) \right) \\ &\leq 2. \end{aligned}$$

However, (13) and the fact that the sum (12) is > 1 imply that relabelling the two groups $g(a,1)$ and $g(a,2)$ would increase the sum (8), contradicting the maximality of (8). □

The above result can be extended in a "straightforward" way to show the following corollary.

COROLLARY 4.2.

(i) $m(2^k, r) \leq 3^k$ *for any positive integer* $k \geq 1$.

(ii) $m(n,r) \leq 3^{\lceil \log_2 n \rceil}$.

Basically, for part (i) we can induct on k, and part (ii) follows from (i). This extended result gives good bounds when n is small. In fact, we can also show results such as $m(3,r) \leq 6$ by the same idea, with more tedious analysis. Since these results are not generally good and the arguments, though intuitively simple, are too tedious to present, we omit their proofs here.

5. The new upper bounds. Next, we give a coloring algorithm yielding a general upper bound which is good for small values of r. The new upper bound implies a stronger value than the conjectured value of $2n-1$ when $r = 2$.

THEOREM 5.1. *When* $n, r \geq 2$, *we have*

$$m(n,r) \leq \left\lceil \left(\frac{r+1}{2} \right) n \right\rceil.$$

Proof. Consider $G = (A, B; E) \in \mathcal{B}_r^n$. Recall that for each $v \in V = A \cup B$, we use $I(v)$ to denote the set of edges incident to v and $g(v,i)$ to denote the set of edges in group i of v. Now, for each vertex $u \in A$ (respectively, B) and each vertex $v \in B$ (respectively, A), define n sets of edges $S_u(v,i)$ as follows:

$$S_u(v,i) = g(u,i) \cap I(v), \quad i = 1, \ldots, n. \tag{14}$$

In other words, $S_u(v,i)$ is the set of edges in group i of u which are incident to v. Let $w_u(v,i)$ be the total weight of edges in $S_u(v,i)$. (We set $w_u(v,i) = 0$ if $S_u(v,i) = \emptyset$.)

Then the grouping condition on G implies that

$$\sum_{b\in B} w_a(b,i) \le 1 \quad \forall a \in A, i = 1,\dots,n, \tag{15}$$

$$\sum_{a\in A} w_b(a,i) \le 1 \quad \forall b \in B, i = 1,\dots,n. \tag{16}$$

To this end, for each $u \in A$ (respectively, B) and each $v \in B$ (respectively, A), let $L_u(v)$ be the set of group names i, $1 \le i \le n$, for which $w_u(v,i) > 1/2$, and let $\bar{L}_u(v)$ be the set of the rest of the indices. More formally,

$$L_u(v) = \{i \mid w_u(v,i) > 1/2, i = 1,\dots,n\}, \tag{17}$$

$$\bar{L}_u(v) = \{1,\dots,n\} - L_u(v). \tag{18}$$

Due to (15), for each index i and a particular vertex $a \in A$, there can be at most one $b \in B$ where $w_a(b,i) > 1/2$. Hence, for each $a \in A$ we must have

$$\sum_{b\in B} |L_a(b)| \le n. \tag{19}$$

Similarly, due to (16), for each $b \in B$ the following holds:

$$\sum_{a\in A} |L_b(a)| \le n. \tag{20}$$

Now, define a weighted bipartite multigraph $G' = (A, B; E')$ as follows.

- For each $a \in A$ and $b \in B$, there are n edges between a and b in G', denoted by $e(a,b,i)$, $1 \le i \le n$. The weight of $e(a,b,i)$, denoted by $w'(a,b,i)$, is defined below. Note that G' is rn-regular.
- For each $a \in A$ and $b \in B$, if $|L_a(b)| \le |L_b(a)|$, then

$$w'(a,b,i) = w_a(b,i), \quad i = 1,\dots,n.$$

Otherwise, when $|L_a(b)| > |L_b(a)|$ define

$$w'(a,b,i) = w_b(a,i), \quad i = 1,\dots,n.$$

First, we claim that any valid coloring of G' induces a valid coloring of G. The term "valid coloring" here means that the total weight of same color edges which are incident to a particular vertex of G' is at most 1. To see this, suppose we are given a valid coloring of G' where the edge $e(a,b,i)$ is colored $c(a,b,i)$, say. Then when $|L_a(b)| \le |L_b(a)|$ we color all edges in the set $S_a(b,i)$ with color $c(a,b,i)$. On the other hand, when $|L_a(b)| > |L_b(a)|$ the set $S_b(a,i)$ gets the color instead.

To this end, let H be the spanning bipartite subgraph of G' obtained from G' by taking only edges whose weights are $> 1/2$. We claim that H has maximum degree at most n. To see this, consider any vertex $a \in A$ of H. We have

$$deg_H(a) = \sum_{b\in B} \min\{|L_a(b)|, |L_b(a)|\} \le \sum_{b\in B} L_a(b) \le n, \tag{21}$$

by (19). Similarly, $deg_H(b) \le n$ for all $b \in B$. Add more edges of G' into H so that H is n-regular. This is possible since G' has n parallel edges between any pair

$(a, b) \in A \times B$. König's line coloring theorem [12] implies that H is n-edge-colorable. (The actual coloring algorithms can be found in [4, 7, 8], for instance.) The graph $G' - E(H)$ is $(r-1)n$-regular; hence it is $(r-1)n$-edge-colorable. However, each edge of $G' - E(H)$ has weight at most $1/2$; hence every two colors can be combined into one without violating the condition that the total weight of same color edges at each vertex is at most 1. Consequently, we can color edges of G' with

$$n + \left\lceil \left(\frac{r-1}{2} \right) n \right\rceil = \left\lceil \left(\frac{r+1}{2} \right) n \right\rceil$$

colors. □

Note that this theorem gives the best upper bounds so far for $m(n, r)$ when r is small, as formally put in the following corollary.

COROLLARY 5.2. *When $n \geq 2$, we have*

(i) $m(n, 2) \leq \lceil \frac{3n}{2} \rceil$,
(ii) $m(n, 3) \leq 2n$,
(iii) $m(n, 4) \leq \lceil \frac{5n}{2} \rceil$.

The argument given in Theorem 5.1 can be extended easily to show the following corollary, whose proof we omit.

COROLLARY 5.3. *The general 3-stage Clos network $C(n_1, r_1, m, n_2, r_2)$ is multirate rearrangeable when*

$$m \geq \frac{(r+1)n}{2},$$

where $n = \max\{n_1, n_2\}$, and $r = \max\{r_1, r_2\}$.

Theorem 3.1 and part (i) of Corollary 5.2 imply $5n/4 \leq m(n, 2) \leq 6n/4$. Given that the number $5/4$ is somewhat "ugly," we make the following conjecture.

CONJECTURE 5.4.

$$m(n, 2) = \left\lfloor \frac{3n}{2} \right\rfloor, \ n \geq 2.$$

In fact, recalling $m(2, r) = 3$, it is very tempting to also make the following conjecture.

CONJECTURE 5.5. *The symmetric 3-stage Clos network $C(n, m, r)$ is multirate rearrangeable if there are at least*

$$\left\lfloor \left(1 + \frac{1}{2} + \cdots + \frac{1}{2^{r-1}} \right) n \right\rfloor = \left\lfloor 2n \left(1 - \frac{1}{2^r} \right) \right\rfloor$$

middle-stage switches. In other words,

$$m(n, r) \leq \left\lfloor 2n \left(1 - \frac{1}{2^r} \right) \right\rfloor.$$

We believe that the upper bound is also the exact value for $m(n, r)$. However, as there is no rigorous evidence yet, we have conjectured a weaker result. Next, we give another upper bound which beats all existing bounds when r is relatively small compared to n.

THEOREM 5.6. *When $n, r \geq 2$, we have*

$$m(n, r) \leq 2n - 1 + \left\lceil \frac{r-1}{2} \right\rceil. \tag{22}$$

Proof. Consider $G = (A, B; E) \in \mathcal{B}_r^n$. Suppose e and e' are two edges connecting two vertices $a \in A$ and $b \in B$, with $w(e) + w(e') \leq 1$. Create a new graph G' from G by collapsing e and e' into one edge with weight $w(e) + w(e')$. Then a valid coloring of G' induces a valid coloring of G.

Now, for every pair $(a, b) \in A \times B$, as long as there are two edges e and e' between a and b for which $w(e) + w(e') \leq 1$, collapse e and e' into one as described. After this procedure is finished, between any pair a and b there is at most one edge with weight $\leq 1/2$, and the rest have weights $> 1/2$. Let H be the resulting graph. Call the edges of H with weight $> 1/2$ *heavy* and the rest of the edges *light*. Since the total weight of edges incident to each vertex of G is at most n, every vertex of H is incident to at most $2n - 1$ heavy edges. In other words, the heavy degree of any vertex of H is at most $2n - 1$.

We claim that the light degree of any vertex of H is at most $r - 1$. To see this, consider $a \in A$. If the heavy degree of A is $2n-1$, then no light edge incident to a can share the same neighbor as a heavy edge of a. Suppose, on the contrary, that there is a heavy edge e and a light edge e', both of which connect a and b. Then the total weight of the other $2n-2$ heavy edges of a except e is $> n-1$; hence $w(e)+w(e') < 1$, as the total weight associated with a is at most n. Consequently, e and e' must have been collapsed by our procedure. Thus, the light degree of a is at most $r-1$. Now, if the heavy degree of a is at most $2n - 2$, then there is also a vertex $b \in B$ with heavy degree at most $2n - 2$. If there was no light edge between a and b, then the light degree of a is at most $r - 1$. If there was one light edge between a and b, relabel this light edge "heavy," which does not change the fact that the maximum heavy degree of H is at most $2n - 1$. Again, the light degree of a is now at most $r - 1$.

König's line coloring theorem [12] implies that we can use at most $2n-1$ colors to color the heavy edges of H and at most $r - 1$ colors to color the light edges of H. As the light edges have weights $\leq 1/2$, every two colors of $r - 1$ colors can be combined into one, for a total of at most $2n - 1 + \lceil (r - 1)/2 \rceil$ colors as desired. (Again, the actual coloring algorithms can be found in [4, 7, 8].) □

As we have mentioned, the new bound is good when r is relatively small. This is formally put in the following corollary.

COROLLARY 5.7. *When* $r \leq \frac{n}{2^{k-1}} + 1$, *we have*

$$m(n, r) \leq 2n - 1 + \left\lceil \frac{n}{2^k} \right\rceil.$$

For example, if $r \leq n + 1$, the Clos network $C(n, m, r)$ is multirate rearrangeable with at most $\lceil 5n/2 \rceil - 1$ middle-stage switches; when $r \leq n/4 + 1$, we need only about $17n/8 - 1$ middle-stage switches, and so on .... The argument given in Theorem 5.6 generalizes straightforwardly to the general Clos network case. Hence, we get the following result.

COROLLARY 5.8. *The general 3-stage Clos network* $C(n_1, r_1, m, n_2, r_2)$ *is multirate rearrangeable when*

$$m \geq 2n - 1 + \left\lceil \frac{r - 1}{2} \right\rceil,$$

where $n = \max\{n_1, n_2\}$, *and* $r = \max\{r_1, r_2\}$.

REFERENCES

[1] V. E. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Math Sci. Engrg. 17, Academic Press, New York, 1965.

[2] J. D. Carpinelli and A. Y. Oruç, *Applications of matching and edge-coloring algorithms to routing in Clos networks*, Networks, 24 (1994), pp. 319–326.

[3] S.-P. Chung and K. W. Ross, *On nonblocking multirate interconnection networks*, SIAM J. Comput., 20 (1991), pp. 726–736.

[4] R. Cole and J. Hopcroft, *On edge coloring bipartite graphs*, SIAM J. Comput., 11 (1982), pp. 540–546.

[5] D. Z. Du, B. Gao, F. K. Hwang, and J. H. Kim, *On multirate rearrangeable Clos networks*, SIAM J. Comput., 28 (1998), pp. 463–470.

[6] P. Fishburn, F. K. Hwang, D. Z. Du, and B. Gao, *On* 1*-rate wide-sense nonblocking for* 3*-stage Clos networks*, Discrete Appl. Math., 78 (1997), pp. 75–87.

[7] H. N. Gabow, *Using Euler partitions to edge color bipartite multigraphs*, Internat. J. Comput. Information Sci., 5 (1976), pp. 345–355.

[8] H. N. Gabow and O. Kariv, *Algorithms for edge coloring bipartite graphs and multigraphs*, SIAM J. Comput., 11 (1982), pp. 117–129.

[9] B. Gao and F. K. Hwang, *Wide-sense nonblocking for multirate* 3*-stage Clos networks*, Theoret. Comput. Sci., 182 (1997), pp. 171–182.

[10] X.-D. Hu, X.-H. Jia, D.-Z. Du, and F. K. Hwang, *Monotone routing in multirate rearrangeable clos networks*, J. Parallel Distrib. Comput., 61 (2001), pp. 1382–1388.

[11] F. K. Hwang, *Rearrangeability of multiconnection three-stage Clos networks*, Networks, 2 (1972), pp. 301–306.

[12] D. König, *Über graphen und ihre anwendung auf determinantentheorie und mengenlehre*, Math. Ann., 77 (1916), pp. 453–465.

[13] T. T. Lee and P. P. To, *Non-blocking routing properties of Clos networks*, in Advances in Switching Networks (Princeton, NJ, 1997), AMS, Providence, RI, 1998, pp. 181–195.

[14] G.-H. Lin, D.-Z. Du, X.-D. Hu, and G. Xue, *On rearrangeability of multirate Clos networks*, SIAM J. Comput., 28 (1999), pp. 1225–1231.

[15] G.-H. Lin, D.-Z. Du, W. Wu, and K. Yoo, *On* 3*-rate rearrangeability of Clos networks*, in Advances in Switching Networks (Princeton, NJ, 1997), AMS, Providence, RI, 1998, pp. 315–333.

[16] R. Melen and J. S. Turner, *Nonblocking multirate networks*, SIAM J. Comput., 18 (1989), pp. 301–313.

[17] H. Q. Ngo, *A new routing algorithm for multirate rearrangeable Clos networks*, Theoret. Comput. Sci., 290 (2003), pp. 2157–2167.

[18] H. Q. Ngo and D. Pan, *WDM Split Cross-Connects and* 3*-Stage Clos Networks*, preprint.

[19] H. Q. Ngo and V. H. Vu, *Multirate rearrangeable Clos networks and a generalized bipartite graph edge coloring problem*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, ACM, New York, 2003, pp. 834–840.

[20] A. Rasala and G. Wilfong, *Strictly non-blocking WDM cross-connects*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, ACM, New York, 2000, pp. 606–615.

[21] A. Rasala and G. Wilfong, *Strictly non-blocking WDM cross-connects for heterogeneous networks*, in Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, Portland, OR, 2000, ACM, New York, 2000, pp. 513–524.

[22] K.-H. Tsai and D.-W. Wang, *Lower bounds for wide-sense non-blocking Clos network*, in Computing and Combinatorics (Taipei, 1998), Springer, Berlin, 1998, pp. 213–218.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 1050–1055

AN APPROXIMATION ALGORITHM FOR THE MINIMUM-COST k-VERTEX CONNECTED SUBGRAPH*

JOSEPH CHERIYAN†, SANTOSH VEMPALA‡, AND ADRIAN VETTA§

Abstract. We present an approximation algorithm for the problem of finding a minimum-cost k-vertex connected spanning subgraph, assuming that the number of vertices is at least $6k^2$. The approximation guarantee is six times the kth harmonic number (which is $O(\log k)$), and this is also an upper bound on the integrality ratio for a standard linear programming relaxation.

Key words. approximation algorithm, ℓ-critically k-connected graph, linear programming relaxation, network design, k-outconnected graph, vertex connectivity

AMS subject classifications. Primary, 68W25, 05C40; Secondary, 68R10, 90C27

DOI. 10.1137/S0097539701392287

1. Introduction.

Let $G = (V, E)$ be an undirected graph, let each edge $e \in E$ have a nonnegative cost c_e, and let k be a positive integer. The *minimum-cost k-vertex connected spanning subgraph (mincost k-VCSS)* problem is to find a spanning subgraph H of minimum cost such that H is k-vertex connected. (A graph is called *k-vertex connected* if it has at least $k+1$ vertices, and the removal of any $k-1$ vertices leaves a connected graph.) The problem is NP-hard for $k \geq 2$, and for $k = 1$ it is the minimum spanning tree problem. Our paper addresses the "special case" of the problem where the graph has order $|V| \geq 6k^2$; this too is NP-hard for $k \geq 2$. (So for a fixed k, our method handles all graphs except a finite set of "small" graphs, and our method fails on each of the "small" graphs.) Our approximation guarantee is six times the kth harmonic number, which is $O(\log k)$. Also, this is an upper bound on the integrality ratio for a standard linear programming relaxation. Several previous papers have attacked the mincost k-VCSS problem (without restrictions on $|V|$), with the goal of improving on the approximation guarantee (see the references). An approximation guarantee of more than $k/2$ has been presented in [11]; also, an upper bound of $O(k)$ on the integrality ratio was known [4, 5]. Better results were not known for our "special case," but we mention that our results may not improve on previous results for small k ($k = 2, 3, 4, \ldots$). (An $O(\log k)$ approximation guarantee was claimed earlier in [15], but subsequently an error has been found, and that claim has been withdrawn; see [16].) For more discussion on related problems and results, see the introduction of [3].

Our algorithm is based on two results: (1) a polynomial-time algorithm of Frank and Tardos [5] for finding a minimum-cost k-outconnected subdigraph of a digraph

*Received by the editors July 12, 2001; accepted for publication (in revised form) March 19, 2003; published electronically July 8, 2003.
http://www.siam.org/journals/sicomp/32-4/39228.html

†Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1 (jcheriyan@math.uwaterloo.ca). This author's work was supported in part by NSERC research grant OGP0138432.

‡Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139 (vempala@math.mit.edu). This author's work was supported in part by NSF Career Award CCR-9875024.

§Department of Computer Science, McGill University, 3480 University Street, Montreal, Quebec, Canada, H3A 2A7 (vetta@cs.mcgill.ca). This author's work was supported in part by NSF Career Award CCR-9875024.

(directed graph), and (2) an upper bound on the order of 3-critical graphs by Mader [12]. The Frank–Tardos algorithm has been applied earlier to the mincost k-VCSS problem by several authors, starting with Khuller and Raghavachari [10]; see also [1, 2, 11]. The scaling trick used in Lemma 3.2 below has been used earlier by [8, 9].

2. Notation and preliminary results. Throughout, we assume that the input graph $G = (V, E)$ is k-vertex connected. Let n denote $|V|$.

2.1. A linear programming relaxation. Let H^* denote a k-VCSS of minimum cost, and let $z^* = c(H^*) = \sum_{e \in E(H^*)} c_e$ denote its cost. The following linear program (LP) $P(k)$ gives a lower bound $z(k)$ on z^* (Frank discusses this LP in [4]). There is a variable x_e, $0 \le x_e \le 1$, for each edge e in G. The intention is that the edge incidence vector of every k-VCSS H (possibly $H = H^*$) forms a feasible solution for $P(k)$. A setpair $W = (W_t, W_h)$ is an ordered pair of disjoint vertex sets, so $W_t \subseteq V$, $W_h \subseteq V$, and $W_t \cap W_h = \emptyset$. An edge uv of G is said to *cover* W if $u \in W_t, v \in W_h$ or $v \in W_t, u \in W_h$. Let $\delta(W)$ denote the set of all edges in G that cover W. If W_t contains at least one vertex, say, p, and W_h contains at least one vertex, say, q, then note that H has at least $k - (n - |W_t \cup W_h|)$ edges in $\delta(W)$, because on removing the vertices in $V - (W_t \cup W_h)$ from H, the resulting graph has at least this number of openly disjoint paths between p and q, and each of these paths contributes one (or more) distinct edges to $\delta(W)$. (Two paths are called *openly disjoint* if every vertex that belongs to both paths is an end vertex of both paths.) Let $\mathcal{S}$ denote the set of all setpairs (W_t, W_h) such that $W_t \ne \emptyset$ and $W_h \ne \emptyset$. It is convenient to keep a parameter ℓ, where ℓ is a positive integer, and write the LP relaxation $P(\ell)$ for the mincost ℓ-VCSS problem.

$$
\begin{aligned}
P(\ell): \qquad z(\ell) \;=\; \text{minimize} \quad & \sum_{e \in E} c_e\, x_e \\
\text{subject to} \quad & \sum_{e \in \delta(W)} x_e \ge \ell \;-\; (n - |W_t \cup W_h|) && \forall\, W \in \mathcal{S}, \\
& 0 \;\le\; x_e \le 1 && \forall\, e \in E.
\end{aligned}
$$

LEMMA 2.1. *Let $z^*(\ell)$ be the minimum cost of an ℓ-VCSS. Then $z^*(\ell) \ge z(\ell)$.*

2.2. k-outconnected subgraphs. A graph is said to be *k-outconnected* from a so-called *root* vertex r if there exist k openly disjoint paths from r to v for each vertex v, $v \ne r$. The *mincost k-OC* problem is as follows: given an undirected graph $G = (V, E)$, a root vertex $r \in V$, and nonnegative costs on the edges, find a minimum-cost subgraph H of G such that H is k-outconnected from r. This problem is NP-hard for $k \ge 2$.

THEOREM 2.2 (see [5, 10]). *Let $G = (V, E)$, r, and $c : E \to \mathbb{R}_+$ be as above. There is a 2-approximation algorithm for the mincost k-OC problem. Moreover, the subgraph found by this algorithm has cost at most $2z(k)$.*

Proof. In the directed version $\widehat{G}$ of G, each edge e of G is replaced by two oppositely oriented arcs, and each of these two arcs has cost c_e. Here is an LP relaxation (in fact, an LP formulation) $\widehat{P}$ of the directed mincost k-OC problem on $\widehat{G}$ (with any vertex r as the root): There is a variable x_a for each arc a in $\widehat{G}$; let $\mathcal{R}$ denote the set of all setpairs $W = (W_t, W_h)$ such that the root r is in W_t and $W_h \ne \emptyset$; and for $W \in \mathcal{R}$ let $\hat{\delta}(W)$ denote the set of arcs (u, v) in $\widehat{G}$ with $u \in W_t$, $v \in W_h$.

$$\widehat{P}: \quad \text{minimize} \quad \sum_{a \in E(\widehat{G})} c_a\, x_a$$

$$\text{subject to} \quad \sum_{a \in \hat{\delta}(W)} x_a \geq k \;-\; (n - |W_t \cup W_h|) \qquad \forall\, W \in \mathcal{R},$$

$$0 \;\leq\; x_a \leq 1 \qquad \forall\, a \in E(\widehat{G}).$$

This LP $\widehat{P}$ has an integer optimal solution (see [4, Theorems 2.1, 2.2]). The Frank–Tardos algorithm solves the directed mincost k-OC problem on $\widehat{G}$ by finding a minimum-cost subdigraph $\widehat{H}$ that is k-outconnected from r, and the cost $c(\widehat{H})$ equals the optimal value of $\widehat{P}$. (The arc incidence vector of $\widehat{H}$ forms an optimal solution of $\widehat{P}$.) Finally, we claim that the optimal value of $\widehat{P}$ is at most $2z(k)$; hence the undirected version of $\widehat{H}$ satisfies the theorem. (It is a subgraph of G that is k-outconnected from r, and it has cost at most $2z(k)$.)

To see that the optimal value of $\widehat{P}$ is at most $2z(k)$, observe that the LP relaxation of the directed mincost k-VCSS problem on $\widehat{G}$ has optimal value at most $2z(k)$ (because a feasible solution $\boldsymbol{x}$ of $P(k)$ (the k-VCSS LP on G) gives a feasible solution of the directed k-VCSS LP on $\widehat{G}$ by assigning the value x_e to each of the two arcs corresponding to each edge e). Moreover, an optimal solution of the directed k-VCSS LP on $\widehat{G}$ is also a feasible solution of $\widehat{P}$. Our claim follows. □

Remark. Our algorithm may apply this result to find a solution to the mincost ℓ-OC problem that has cost at most $2z(\ell)$, where $1 \leq \ell \leq k$.

2.3. 3-critical graphs. For a graph G, let $\kappa(G)$ denote the *vertex connectivity*, i.e., the minimum number of vertices whose removal results in a disconnected graph or the trivial graph (namely, K_1). An *ℓ-separator* of a connected graph is a set of ℓ vertices whose removal results in a disconnected graph.

A graph $G = (V, E)$ is called 3-*critical* if the vertex connectivity decreases by $|S|$ on removing the vertices in any set S of at most three vertices, that is, if $\kappa(G - S) = \kappa(G) - |S|$ for all $S \subseteq V$, $|S| \leq 3$. If G is *not* 3-critical, then note that there exists a set S of three vertices such that no $\kappa(G)$-separator contains all the vertices in S. Mader gives an upper bound on the order of 3-critical graphs [12]. (The proof is written in German, and the result is discussed (without proof) in two survey papers written in English [13, 14].)

THEOREM 2.3 (see Mader [12]). *A 3-critical graph with vertex connectivity k has fewer than $6k^2$ vertices.*

3. The algorithm and its analysis. The algorithm starts with $i := 1$ and a minimum-cost spanning tree H_1. Each iteration $i = 1, 2, \ldots$, augments H_i to H_{i+1} by adding edges from $E(G) - E(H_i)$ such that the vertex connectivity increases by at least one, and the "augmenting cost" $c(H_{i+1}) - c(H_i)$ is approximately minimum. A detailed description of an iteration follows. Let $\ell = \kappa(H_i)$. If $\ell = k$, then we stop and output H_i as the desired k-VCSS. Now, suppose $\ell < k$. For each edge in H_i, we change the cost to zero (the other edges keep the original costs). By Mader's theorem (and the fact that n is at least $6k^2$), there exist three vertices such that no ℓ-separator of H_i contains all three vertices. We find three such vertices r_1, r_2, r_3 by exhaustively checking for each vertex set S of cardinality three whether $\kappa(H_i - S) > \ell - 3$. For each of these three vertices, we apply the Frank–Tardos algorithm with root r_j ($j = 1$, 2, or 3) and the modified edge costs to find a supergraph $H_{i,j}$ of H_i that is $(\ell + 1)$-outconnected from r_j. We take (the edge set of) H_{i+1} to be the union of (the edge

sets of) $H_{i,1}$, $H_{i,2}$, and $H_{i,3}$.

LEMMA 3.1. *At every iteration $i = 1, 2, \ldots$, we have $\kappa(H_{i+1}) \geq \kappa(H_i) + 1$.*

Proof. Let $\ell = \kappa(H_i)$. Note that $\ell < k$. Suppose that $\kappa(H_{i+1}) = \ell$. Then H_{i+1} has an ℓ-separator C, $C \subset V$. Now, H_i is not 3-critical by Mader's theorem, since $n \geq 6k^2 > 6\ell^2$. Hence, there exist three vertices in H_i such that for each ℓ-separator of H_i, at least one of these three vertices is absent from the ℓ-separator. The algorithm finds three such vertices r_1, r_2, r_3. Without loss of generality, r_1 is absent from C. The graph $H_{i,1}$, which is a subgraph of H_{i+1}, is $(\ell+1)$-outconnected from r_1. Hence H_{i+1} has $(\ell+1)$ openly disjoint paths between r_1 and v, for every other vertex v, and one of these paths survives in $H_{i+1} - C$. We have a contradiction, since $H_{i+1} - C$ is connected. The lemma follows. □

LEMMA 3.2. *At every iteration $i = 1, 2, \ldots$, we have $c(H_{i+1}) - c(H_i) \leq \frac{6z(k)}{k-\ell}$, where $\ell = \kappa(H_i)$.*

Proof. Note that $\ell < k$. We will prove that for each of the three supergraphs $H_{i,j}$ ($j = 1$, 2, or 3) of H_i, the augmenting cost $c(H_{i,j}) - c(H_i)$ is at most $2z(k)/(k-\ell)$. Then the lemma follows immediately.

Let $\boldsymbol{x} : E \to \mathbb{R}_+$ be an optimal solution to the LP $P(k)$; note that the cost of $\boldsymbol{x}$ (with respect to the original edge costs c) is $z(k)$.

Recall that (during the construction of $H_{i,j}$, $j = 1, 2, 3$) the edge costs are modified such that an edge already in H_i has zero cost, while the other edges have the original costs. Let $\boldsymbol{x'} : E \to \mathbb{R}_+$ be given by

$$x'_e = \begin{cases} 1 & \text{if } e \text{ is in } H_i, \\ \dfrac{x_e}{k-\ell} & \text{otherwise.} \end{cases}$$

Clearly, $\boldsymbol{x'}$ has modified cost at most $z(k)/(k-\ell)$. We claim that $\boldsymbol{x'}$ is a feasible solution to the LP $P(\ell+1)$. Then, by Theorem 2.2, the Frank–Tardos algorithm finds an $(\ell+1)$-outconnected supergraph of H_i with augmenting cost at most $2z(k)/(k-\ell)$.

To see the claim, consider any setpair $W \in \mathcal{S}$ and its constraint in the LP $P(\ell+1)$,

$$\sum_{e \in \delta(W)} x'_e \geq (\ell+1) - q,$$

where $q = n - |W_t \cup W_h|$. First, suppose that H_i has no edges in $\delta(W)$. Then since H_i is ℓ-vertex connected, we have $q \geq \ell$. If $q \geq \ell + 1$, then, obviously, $\boldsymbol{x'}$ satisfies this constraint. Otherwise, if $q = \ell$, then $\boldsymbol{x'}$ satisfies this constraint because (i) $\boldsymbol{x}$ satisfies the constraint of W in the LP $P(k)$, namely, $\sum_{e \in \delta(W)} x_e \geq k - \ell$, and (ii) each edge $e \in \delta(W)$ has $x'_e = x_e/(k-\ell)$. Now, suppose that H_i has $p \geq 1$ edges in $\delta(W)$. If $p < (\ell+1) - q$, then delete $\leq p$ vertices from W_t and W_h to obtain a new setpair $\hat{W}$ such that $\hat{W}_t \neq \emptyset \neq \hat{W}_h$ and H_i has no edges in $\delta(\hat{W})$, and then apply the previous reasoning to $\hat{W}$ to infer that $\boldsymbol{x'}$ satisfies the constraint of $\hat{W}$ and hence also of W. If $p \geq (\ell+1) - q$, then $\sum_{e \in \delta(W)} x'_e \geq |E(H_i) \cap \delta(W)| = p \geq (\ell+1) - q$. Thus the claim holds. □

THEOREM 3.3. *Suppose that the input graph $G = (V, E)$ is k-vertex connected and has order $|V| \geq 6k^2$. Then the algorithm terminates with a k-VCSS that has cost at most $6(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k})z(k)$, where $z(k)$ is the optimal value of the LP relaxation. The algorithm runs in time $O(k^2 n^4 (n + k^{2.5}))$.*

Proof. The vertex connectivity of H_i increases by at least one in every iteration, starting from one, so the algorithm terminates with a k-VCSS in at most $k - 1$

iterations. The cost of the k-VCSS is

$$\leq c(H_1) + \sum_{i=1}^{k-1}(c(H_{i+1}) - c(H_i)) \leq \frac{2z(k)}{k} + \sum_{\ell=1}^{k-1}\frac{6z(k)}{k-\ell}$$
$$\leq 6\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k}\right) z(k).$$

(Note that the minimum spanning tree H_1 is an optimal solution to the mincost 1-OC problem (with any vertex as the root), and an optimal solution $\boldsymbol{x}$ of the LP $P(k)$ gives a feasible solution $\frac{1}{k}\boldsymbol{x}$ of the LP $P(1)$; hence by the proof of Theorem 2.2, $c(H_1) \leq 2z(1) \leq \frac{2z(k)}{k}$.)

To see the running time, note that each iteration i $(1 \leq i < k)$ runs the Frank–Tardos algorithm at most three times and tests $\kappa(H_i - S)$ for at most n^3 sets of vertices S of cardinality three. Gabow's algorithm [7] tests the vertex connectivity κ in time $O((n + \kappa^{2.5}) \cdot \kappa n)$, and there is a version of the Frank–Tardos algorithm, due to Gabow, that runs in time $O(k^2 n^2 |E|)$ [6, Theorem 4.5]. □

Acknowledgment. We thank Zeev Nutov for several discussions.

REFERENCES

[1] V. Auletta, Y. Dinitz, Z. Nutov, and D. Parente, *A 2-approximation algorithm for finding an optimum 3-vertex-connected spanning subgraph*, J. Algorithms, 32 (1999), pp. 21–30.
[2] J. Cheriyan, T. Jordán, and Z. Nutov, *On rooted node-connectivity problems*, Algorithmica, 30 (2001), pp. 353–375.
[3] J. Cheriyan, S. Vempala, and A. Vetta, *Approximation algorithms for minimum-cost k-vertex connected subgraphs*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 306–312.
[4] A. Frank, *Connectivity augmentation problems in network design*, in Mathematical Programming: State of the Art 1994, J. R. Birge and K. G. Murty, eds., The University of Michigan, Ann Arbor, MI, 1994, pp. 34–63.
[5] A. Frank and É. Tardos, *An application of submodular flows*, Linear Algebra Appl., 114/115 (1989), pp. 329–348.
[6] H. N. Gabow, *A representation for crossing set families with applications to submodular flow problems*, in Proceedings of the 4th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1993, pp. 202–211.
[7] H. N. Gabow, *Using expander graphs to find vertex connectivity*, in Proceedings of the 41st IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2000, pp. 410–420.
[8] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, É. Tardos, and D. P. Williamson, *Improved approximation algorithms for network design problems*, in Proceedings of the 5th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1994, pp. 223–232.
[9] K. Jain, I. Mandoiu, V. V. Vazirani, and D. P. Williamson, *A primal-dual schema based approximation algorithm for the element connectivity problem*, in Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1999, pp. 484–489.
[10] S. Khuller and B. Raghavachari, *Improved approximation algorithms for uniform connectivity problems*, J. Algorithms, 21 (1996), pp. 434–450.
[11] G. Kortsarz and Z. Nutov, *Approximating node connectivity problems via set covers*, in Approximation Algorithms for Combinatorial Optimization, Lecture Notes in Comput. Sci. 1913, K. Jansen and S. Khuller, eds., Springer-Verlag, Berlin, 2000, pp. 194–205.
[12] W. Mader, *Endlichkeitsätze für k-kritische Graphen*, Math. Ann., 229 (1977), pp. 143–153.
[13] W. Mader, *Connectivity and edge-connectivity in finite graphs*, in Surveys in Combinatorics (Proc. Seventh British Combinatorial Conf., Cambridge, 1979), London Math. Soc. Lecture Note Ser. 38, Cambridge University Press, Cambridge, UK, 1979, pp. 66–95.

[14] W. Mader, *On k-critically n-connected graphs*, in Progress in Graph Theory, Academic Press, Toronto, Ontario, Canada, 1984, pp. 389–398.

[15] R. Ravi and D. P. Williamson, *An approximation algorithm for minimum-cost vertex-connectivity problems*, Algorithmica, 18 (1997), pp. 21–43.

[16] R. Ravi and D. P. Williamson, *Erratum: "An approximation algorithm for minimum-cost vertex-connectivity problems,"* Algorithmica, 34 (2002), pp. 98–107.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 1056–1066

TREE PATTERN MATCHING TO SUBSET MATCHING IN LINEAR TIME*

RICHARD COLE† AND RAMESH HARIHARAN‡

Abstract. In this paper, we show an $O(n+m)$ time Turing reduction from the tree pattern matching problem to another problem called the *subset matching* problem. Subsequent works have given efficient deterministic and randomized algorithms for the subset matching problem. Together, these works yield an $O\left(n\log^2 m+m\right)$ time deterministic algorithm and an $O(n\log n+m)$ time Monte Carlo algorithm for the tree pattern matching problem.

Key words. tree pattern matching, subset matching

AMS subject classifications. 68W05, 68W40

DOI. 10.1137/S0097539700382704

1. Introduction.

In the tree pattern matching problem, the text and the pattern are ordered, binary trees, and all occurrences of the pattern in the text are sought. Here, the pattern occurs at a particular text position if placing the pattern with root at that text position leads to a situation in which each pattern node overlaps some text node. This problem has a number of applications (see [6]). Actually, in these applications, the tree need not be binary and the edges may be labelled; however, as shown in [4], this general problem can be converted to a problem on binary trees with unlabelled edges but with a blow-up in size proportional to the logarithm of the size of the pattern. In fact, this blow-up can also be avoided in our approach, as we will indicate in our description.

The naive algorithm for tree pattern matching takes time $O(nm)$, where n is the text size and m is the pattern size. Hoffman and O'Donell [6] gave another algorithm with the same worst case bound. This algorithm decomposes the pattern into strings, each string representing a root-to-leaf path. It then finds all occurrences of each of these strings in the text tree. The first $o(nm)$ algorithm was obtained by Kosaraju [9], who first noticed the connection of the tree pattern matching problem to the problem of string matching with don't-cares and the problem of convolving two strings. Kosaraju's algorithm takes $O(nm^{.75}\log m)$ time. Dubiner, Galil, and Magen [4] improved Kosaraju's algorithm by discovering and exploiting periodicities in paths in the pattern. They obtained a bound of $O(nm^{.5}\log m)$. This was the best bound known to date. Dubiner, Galil, and Magen also made the observation that the naive algorithm actually takes $O(nh)$ time, where h is the height of the pattern.

In this paper, we show how to reduce the tree pattern matching problem to the *subset matching* problem in linear time. The subset matching problem is to find all

*Received by the editors March 29, 2000; accepted for publication (in revised form) April 25, 2003; published electronically July 8, 2003. This work was supported in part by NSF grants CCR9202900, CCR9503309, CCR9800085, and CCR0105678. This work is based on an earlier work: "Tree pattern matching and subset matching in randomized $0(n\log^3 m)$ time," in Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing (STOC'97), © ACM, 1997. http://doi.acm.org/10.1145/258533.258553.

http://www.siam.org/journals/sicomp/32-4/38270.html

†Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012-1185 (cole@cs.nyu.edu).

‡Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India (ramesh@csa.iisc.ernet.in). This work was done in part while this author was visiting NYU.

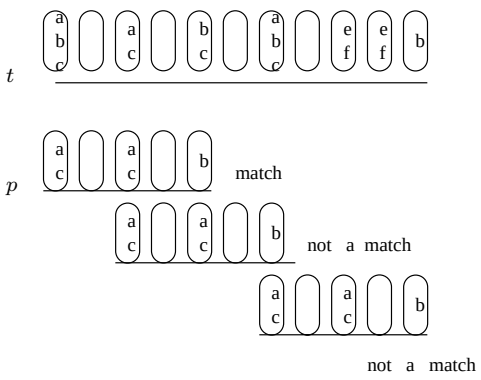


FIG. 1. *Example of subset matching.*

occurrences of a pattern string p of length m in a text string t of length n, where each pattern and text location is a set of characters drawn from some alphabet. The pattern is said to occur at text position i if the set $p[j]$ is a subset of the set $t[i+j-1]$ for all j, $1 \leq j \leq m$. It is required to find all text locations at which the pattern matches; i.e., each pattern set is a subset of the aligned text set (see Figure 1).

The reduction from tree pattern matching to subset matching proceeds in two steps.

• We show that the general tree pattern matching problem can be reduced to the following special case, called *spine pattern matching*, by a linear time Turing reduction. In spine pattern matching, there is a special path in each of the pattern and text called their spines. The spine begins at the root of its tree, and in addition each node on the spine has at most one nonspine child. Spines have additional properties as well, which will be described later. All matches of the pattern in the text are sought with the additional restriction that the spine of the pattern must match a portion of the spine of the text; i.e., nodes on the pattern spine must be aligned with nodes on the text spine. For intuition, one can think of the spine as being the path of left children starting at the root (and in fact one can reduce the general problem to this case in linear time, although we will not do so).

The above reduction may create several instances of the spine pattern matching problem, but the sum of the sizes of these instances will be linear. This reduction is completely deterministic. It proceeds by using the periodicity structure of paths and by decomposing the text tree into periodic paths in a nontrivial manner. Each path then gives a spine for the spine pattern matching problem.

• Next, we reduce the spine pattern matching problem to the subset matching problem in linear time. This is, in fact, readily done. The spine of the text tree gives the text string for the subset matching problem; the subtrees hanging from this spine determine the various text sets. Analogous facts hold for the pattern.

The two reductions above imply that the tree pattern matching problem can be reduced to several instances of the subset matching problem, the sum of the sizes of these instances being linear. Therefore, an algorithm for the subset matching problem yields an algorithm for the tree pattern matching problem with the same time complexity.

Cole and Hariharan [1] gave a randomized algorithm for the subset matching problem running in time $O((n+s)\log^3 m)$, where s is the sum of the sizes of all

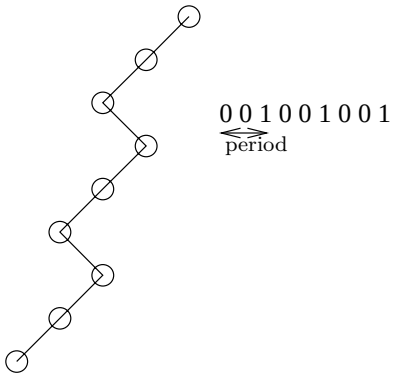


FIG. 2. *A path and its associated string.*

the pattern and text sets. Subsequently, Indyk [7] gave a deterministic algorithm for the subset matching problem running in time $O((n+s)m^{\sqrt{\frac{\log\log m}{\log m}}(1+o(1))})$. Later, Cole, Hariharan, and Indyk [3] gave a deterministic algorithm running in time $O((n+s)\log^3 m)$ and a randomized algorithm running in time $O((n+s)\frac{\log^3 m}{\log\log m})$. Indyk [8] also gave a randomized Monte Carlo algorithm with running time $O(cn\log n)$ and failure probability $O(1/n^c)$ for any fixed constant $c \geq 1$. Finally, Cole and Hariharan [2] gave a deterministic algorithm running in time $O(n\log^2 m)$. It follows that there is a deterministic algorithm running in time $O(n\log^2 m)$ and a Monte Carlo randomized algorithm running in time $O(n\log n)$ for the tree pattern matching problem.

This paper is organized as follows. Section 2 gives some required definitions. Section 3 describes the reduction of the spine pattern matching problem to the subset matching problem. Section 4 describes the reduction from the tree pattern matching problem to the spine pattern matching problem.

2. Definitions.

DEFINITION 2.1 (tree pattern matching). *We consider ordered binary trees; i.e., each internal node has a left and/or a right child. The text tree t has n nodes and the pattern tree p has m nodes. The problem entails finding all nodes v in t where p matches; i.e., when the root of p is aligned with v, each node in p is aligned with a node in t.*

DEFINITION 2.2 (paths, strings, and periods). *Note that paths in trees p and t can be expressed as strings over a two character alphabet, one character signifying a left edge and the other a right edge (see Figure* 2: 0 *represents a left edge and* 1 *a right edge). The* period *of a string $s[1\ldots|s|]$ is the smallest number $j > 0$ such that $s[i] = s[i+j]$ for all i, $1 \leq i \leq |s| - j$. If no such j exists, then the period of s is defined to be $|s|$. The* period *of a path is defined to be the period of its associated string.*

It is well known that the period can be computed in linear time [5]. The following lemma is classical [10].

LEMMA 2.3. *If $k \leq |s| - j$ is such that the period j of s does not divide k, then the string $s[k+1\ldots k+j]$ differs from the string $s[1\ldots j]$.*

DEFINITION 2.4 (spine pattern matching). *This is a restricted version of the tree pattern matching problem. In this problem, the text and the pattern each have one designated path, called their* spines. *The text and pattern spines originate at their*

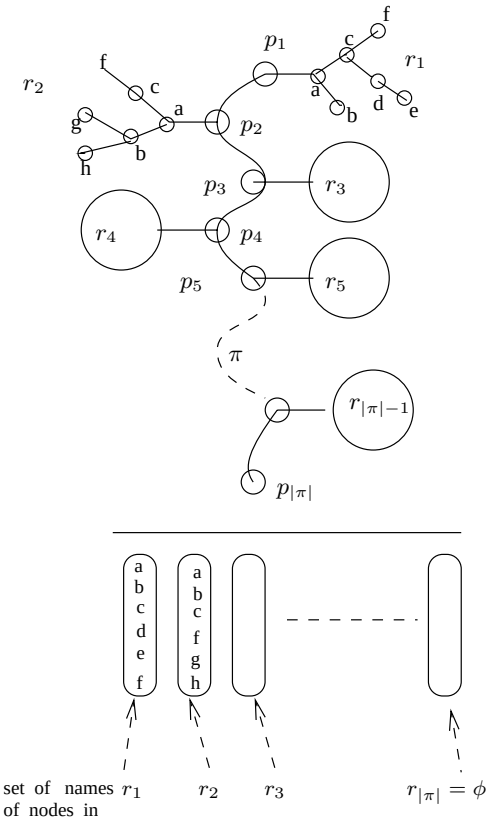


FIG. 3. *The spine and its associated set string.*

respective roots and are maximal paths having the same period, θ say (the θ needed for tree pattern matching will be determined later). If the input does not have this form, it is not a legitimate input for this problem. In fact, both spines when represented as strings will have the form $x^k x'$, where $|x| = \theta$ and x' is a prefix of x (here, the values of k and x' could differ for the pattern spine and the text spine, but x is identical for both spines). All matches of the pattern in which the pattern spine falls completely on the text spine are sought.

From maximality, it follows that both spines terminate at nodes with at most one child (a child which when added to the spine destroys its periodic structure). Since both spines have the same period θ, it follows that the pattern spine will fall completely on the text spine only if the root of the pattern is placed at certain nodes on the text spine. These nodes will occur at integer multiples of θ from the text root and will be designated "anchor nodes."

3. Reducing spine pattern matching to subset matching. The spines of the pattern p and the text t will define the strings for the subset matching problem. The subsets at each location in these strings will correspond to the off-spine subtrees of the spine nodes; an *off-spine subtree* is a subtree whose root is a nonspine node but the parent of whose root is on the spine. These subsets are obtained by labelling the nodes of the off-spine subtrees as follows (see Figure 3). The key fact about this labelling is that two nodes in two distinct off-spine subtrees (both of which could be

in the pattern or in the text, or, alternatively, one could be in the pattern and the other in the text) get the same label if and only if the paths from these nodes to the roots of their respective off-spine subtrees represent identical strings.

The off-spine subtrees of p are labelled first. The subtrees are overlaid to form a *combined pattern subtree*; the overlaying aligns the roots of the off-spine subtrees and recursively overlays their subtrees. Then the combined pattern subtree is traversed by any convenient method, e.g., a breadth first traversal, and the nodes are labelled by the associated numbering. For each spine node, we form a subset consisting of the collection of numbers labelling the nodes of its off-spine subtree. This collection of subsets defines the pattern for the subset matching problem instance. The off-spine subtrees of t are labelled using the same labelling. To do this, each off-spine text subtree and the combined pattern subtree are traversed in lock-step. Consider the text subtree laid over the combined pattern subtree. Clearly, any text node that lies beyond the combined pattern subtree will not be part of any match in which the pattern spine is aligned with a portion of the text spline. Consequently, these text nodes need not be and are not given labels, and indeed need not be and are not traversal. As a result, we have the following easy fact about the time complexity of the above computation.

FACT 1. *The labels to nodes in off-spine subtrees of the pattern can be given in* $O(m)$ *time. The labels to any one off-spine subtree* t' *in the text can be given in time* $O(\min\{|t'|, m\})$. *The total time taken for the labelling is thus* $O(n+m)$*; consequently, the size of the resulting subset matching problem is also* $O(n+m)$.

Recall our remark from the introduction that the case of larger degree and labelled trees can be handled without any extra overhead. Larger degree is simply handled by the usual binarization. Labelled trees are handled by pairing the given labels with the labels obtained here.

Clearly, there is a match in the subset matching problem beginning at a location corresponding to an anchor node if and only if there is a match in the spine pattern matching problem with the pattern tree root aligned with the corresponding anchor node. This completes the reduction from spine pattern matching to subset matching.

4. Reducing tree pattern matching to one or more instances of spine pattern matching. Consider two matches of the pattern with the text in which the pattern instances overlap in $m/2$ or more locations. To avoid checking such matches independently, we seek to have the roots of both pattern instances lie on the same instance of a spine obtained from t.

DEFINITION 4.1. *The* size *of a node* v *in a tree is defined to be the number of nodes in the subtree rooted at* v. *Let* t_v *denote the subtree of* t *rooted at a node* v *in* t, *and let* p_v *denote the subtree of* p *rooted at a node* v *in* p.

4.1. Processing the pattern. We define the spine π of the pattern p to be the following path from the root to a node with at most one child. π consists of two segments, π_1 and π_2. π_1 is a centroid path; i.e., it is obtained by moving to the child with larger size at each step, with ties broken arbitrarily. π_1 ends when a node x such that $|p_x| \leq \frac{m}{2}$ is reached. Note that $|p_x| \geq \frac{m}{4}$. Let θ be the period of π_1. π_2 is the longest path starting at x such that the path π continues to have period θ. Note that π_2 has a vertex in common with π_1. π is readily computed in linear time. π_1 is terminated at x rather than at a node with at most one child to guarantee an overall linear-sized construction.

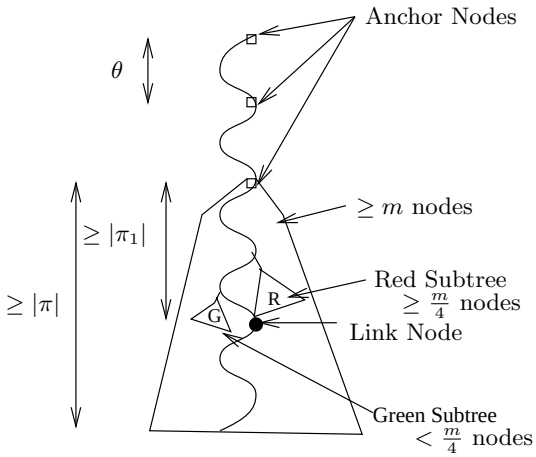


FIG. 4. *A θ-Path in C.*

4.2. Decomposing the text.

DEFINITION 4.2 (see Figure 4). *A path in t from a node u to a node v in t_u is a* θ-path *if it has period θ and is identical to the spine of the pattern in the first θ locations (when both paths are viewed as strings). This path is* maximal *if extending it to the distance θ ancestor of u or either child of v results in a path which is not a θ-path (in fact, v can have only one child). These paths are the candidate spline paths in the text, but we need to impose some further restrictions. Continuing with the definitions, the* link *node l in this path is the node closest to v such that $|t_l| \geq \frac{m}{4}$. An* anchor *node on this path is a node at distance an integer multiple of θ from the start node. The strong* anchor *nodes w on this path also satisfy the following properties. As we will see, matches occur only when the pattern root is aligned with a strong anchor node.*

1. *t_w has at least m nodes.*
2. *The distance from w to l is at least $|\pi_1|$, and thus has length at least θ.*
3. *The distance from w to v is at least $|\pi|$.*
4. *Consider the subtrees hanging from the maximal θ-*path *starting at w. Classify them as* red *subtrees if they have at least $\frac{m}{4}$ nodes and as* green *subtrees otherwise. If all these subtrees except exactly one are green, then the green subtrees plus the path together have at least $m/2$ nodes.*
5. *Either $u = w$ or the distance from u to w is an integer multiple of θ.*

We form a collection C of maximal θ-paths in t, whose start nodes are strong anchor nodes; i.e., they satisfy properties 1–5 above.

Clearly, if any of properties 1–3 or 5 do not hold, there cannot be a match with p's root aligned with w. To see the need for property (4) we argue as follows. Consider a match of p in which at most one of the off-spine subtrees R in t is red. As the spine of p does not match any nodes in R, the subtree of p matching R has size less than $m/2$. Consequently, the remainder of p, of size at least $m/2$, matches the aligned spine portion in t and its green subtrees, which therefore have combined size at least $m/2$.

Note that the paths in C need not be disjoint; however, their combined length will still be $O(n)$, as we shall show later in Lemma 5.18.

The algorithm for constructing these paths is given next.

The path decomposition algorithm. The decomposition is obtained using the following algorithm. For each node x in T, this algorithm first determines the longest θ-path which begins at x. This is done in $O(n)$ time using a Knuth–Morris–Pratt-type automaton in conjunction with a depth-first traversal of t as in the algorithm of Hoffman and O'Donell [6]. Next, the algorithm determines those maximal θ-paths found above which satisfy properties 1–4, discarding all other paths. To this end, it computes the size of each subtree, which allows property 1 to be tested. Properties 2–4 are readily tested by means of a subsequent traversal of each path. It will also be useful to determine, for each such maximal θ-path, whether the node at distance θ from the start of the path is also a strong anchor node. Since, as we will see, the sum of the length of paths in C is $O(n)$, the total time taken above is $O(n)$.

Thus determining matches of p at strong anchor nodes on paths in C suffices to determine all matches of p in t. Further, note that when p is placed with its root at a strong anchor node on some path in C, the spine of p lies completely on that path.

4.3. Processing paths in C. The purpose of processing a path $\rho \in C$ is to determine whether or not p matches at w for each anchor node w on ρ. Each path ρ in C will be processed as follows.

Let u be the node at which ρ starts. u itself is a strong anchor node. Whether or not the pattern matches at u is determined in a brute force manner. This requires $O(m)$ time. We will show in Lemma 5.17 that there are $O(n/m)$ paths, and hence the total time taken over all paths in this process is just $O(n)$.

Matches at other strong anchor nodes on ρ are determined differently, i.e., by reduction to an instance of the spine pattern matching problem.

Consider the portion of ρ starting from the second anchor node onwards, denoted $trunc(\rho)$. If $trunc(\rho)$ starts with a strong anchor node, it provides the spine of the text instance. Clearly, there is a match of p rooted at an anchor node on $trunc(\rho)$ if and only if there is a match at the same location in the corresponding spine pattern matching problem instance.

This concludes the reduction.

5. The analysis. Let $s_1, \ldots, s_{|\rho|-\theta}$ denote the off-spine subtrees, if any, for $trunc(\rho)$, in increasing order of distance from the start node of ρ. Some of the s_i's might not exist. By Fact 1, reducing this instance of the spine pattern matching problem to the subset matching problem takes time $O(\sum_{i=1}^{|\rho|-\theta} \min\{|s_i|, m\} + |\rho| - \theta)$ (plus, of course, $O(m)$ time for processing the pattern, which is common to all the instances of the spine pattern matching problem which result above); also, it yields a text of size $O(\sum_{i=1}^{|\rho|-\theta} \min\{|s_i|, m\} + |\rho| - \theta)$ and a pattern of size $O(m)$.

The total time taken to process ρ is thus $O(m + \sum_{i=1}^{|\rho|-\theta} \min\{|s_i|, m\} + |\rho| - \theta)$. This quantity can be split into four parts: $O(m)$ time for checking for an occurrence of p at the first anchor node, time proportional to its size for each green subtree hanging from $trunc(\rho)$, $O(m)$ time for each red subtree hanging from $trunc(\rho)$, and $O(|\rho| - \theta)$ time for the path itself. We need to show that this sums to $O(n)$ over all paths ρ. By Lemma 5.17, there are $O(n/m)$ paths; hence the first part sums to $O(n)$ time. By Corollary 5.3, the green subtrees in the truncated paths are disjoint; hence the second part sums to $O(n)$. By Lemma 5.8, there are $O(n/m)$ red subtrees; hence the third part sums to $O(n)$. Finally, by Corollary 5.2, the truncated path lengths sum to $O(n)$, and hence the fourth part sums to $O(n)$ also. This yields $O(n)$ time overall. The same argument shows the resulting subset matching problems have texts of total size $O(n)$; also, they each have the same pattern of size $O(m)$.

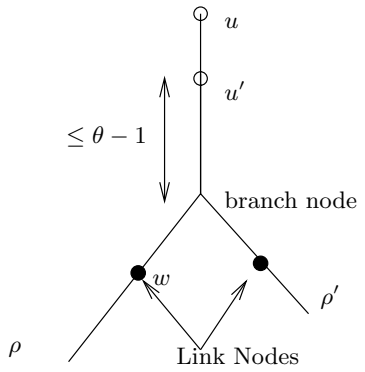


FIG. 5. *Overlap is at most* $\theta - 1$.

5.1. Showing $O(n)$ time.

5.1.1. Some properties of paths in C.

LEMMA 5.1. *Consider two paths* ρ, ρ' *in* C *starting at nodes* u *and* u'*, respectively (see Figure* 5*). Suppose* u' *lies on* ρ*. Then at most the first* $\theta - 1$ *edges of* ρ' *are also present in* ρ*.*

Proof. From the construction of C, the length of the path between u and u' is not divisible by θ. The lemma then follows from Lemma 2.3. ∎

COROLLARY 5.2. *If the first* θ *edges are removed from each path in* C*, then the resulting collection of paths is node disjoint. Hence the truncated paths have total lengths* $O(n)$*. Also, as the link node is not among the first* θ *nodes by property* 2 *of paths in* C*, the link node of* ρ' *cannot lie on* ρ*.*

COROLLARY 5.3. *The green subtrees hanging from the truncated paths are all disjoint.*

Proof. It suffices to consider the green subtrees hanging from two paths $\rho, \rho' \in C$, starting at u, u', respectively, where u is a proper ancestor of u' (for if u and u' are unrelated, then clearly the green trees hanging from ρ and ρ' are disjoint). By Corollary 5.2, $trunc(\rho)$ and $trunc(\rho')$ are disjoint. For a contradiction, suppose that G is a green subtree hanging from $trunc(\rho)$ and containing v, a node in a green subtree hanging from $trunc(\rho')$. It follows from Lemma 5.1 that $trunc(\rho')$ lies within G. But $trunc(\rho')$ includes the first anchor node w' on ρ', and the subtree of t rooted at w' contains at least m nodes. Then G, which contains this subtree, would be red. ∎

LEMMA 5.4. *Let* ρ, ρ' *be as in Lemma* 5.1 *(see Figure* 5*). Then* ρ' *cannot overlap a node in* ρ *which is a proper descendant of* ρ*'s link node* w*. Therefore, if* ρ' *overlaps the link node* w *of* ρ*, then it branches away from* ρ *at* w*.*

Proof. By Corollary 5.2, the link node l' of ρ' is not on ρ. If ρ' overlaps a node w' in ρ which is a proper descendant of w, then $|t_{w'}| \geq |t_{l'}| \geq \frac{m}{4}$, and therefore w cannot be the link node of ρ, a contradiction. ∎

LEMMA 5.5. *Let* ρ*,* ρ'*, and* ρ'' *be three paths whose start nodes appear in this order on the path from the root of* t *to the start node of* ρ''*. Suppose that* ρ'' *overlaps* ρ'*'s link node, and suppose that* ρ' *shares at least one edge with* ρ*. Then* ρ *and* ρ'' *do not overlap.*

Proof. Suppose, for a contradiction, that ρ and ρ'' overlap. Consider the portion q of ρ' up to the start node of ρ''. By assumption, q lies entirely on ρ, and thus by Lemma 5.1, $|q| \leq \theta - 1$. But q is a period of the portion of ρ' from its start node

to its link node, as can be seen by considering the overlap of ρ' with ρ''. Further, this portion of ρ' has π_1 as a prefix, and as $|\pi_1| \geq \theta$, q would also be a period of π_1, contrary to the definition of θ. □

5.1.2. Bounding the number of red subtrees. We bound the number of red subtrees hanging from the truncated paths in C by associating them in part with a set of $O(n/m)$ nodes of t, called *marked* nodes. A path with k red subtrees will have $k-1$ marked nodes.

DEFINITION 5.6. *A node in t is marked if its left and right subtrees both contain at least $\frac{m}{4}$ nodes.*

LEMMA 5.7. *The number of marked nodes in t is $O(\frac{n}{m})$.*

Proof. Each subtree of $m/4$ or more unmarked nodes is contracted to a single unmarked leaf if its parent is marked. Each maximal subtree of unmarked nodes whose parent is unmarked is discarded. Finally, maximal paths of unmarked nodes are replaced by single edges. This reduces the original tree to a new tree in which all internal nodes are marked and correspond one to one to the original marked nodes, and all leaves are unmarked, each corresponding to $m/4$ or more distinct unmarked nodes in t. Further, each internal node has two children. Thus, as there are at most $4n/m$ leaves, there are fewer than $4n/m$ internal nodes. □

LEMMA 5.8. *There are $O(n/m)$ red subtrees hanging from truncated paths whose first node is a strong anchor.*

Proof. Suppose strongly anchored truncated path ρ has $k > 1$ red subtrees hanging from it. Such a path has $k-1$ marked nodes (namely the *lcas* of the red subtree roots). By Corollary 5.3, as these truncated paths are disjoint, and by Lemma 5.7, as there are $O(n/m)$ marked nodes, there can be only $O(n/m)$ anchored truncated paths with two or more red subtrees.

If a strongly anchored truncated path has only one red subtree then by property 4 of the paths, the path together with its green subtrees contains at least $m/2$ nodes. This is also true if it has no red subtrees. By Corollaries 5.2 and 5.3, the paths and their green subtrees are disjoint from each other. Consequently, there are only $O(n/m)$ such paths. □

5.1.3. Bounding the number of paths. It remains to bound the number of paths. To this end, we form suitable collections of paths.

Consider the following procedure for forming collections of paths. A collection starts with a path $\tilde{\rho}$ whose start node is not overlapped by any other path; $\tilde{\rho}$ is called the *root* path of the collection. The collection is built up by iterating the following step. For each path ρ in the collection, every path ρ', whose start node is on ρ but which does not overlap the link node of any other path, is added to the collection. Call these level 1 collections. Level $i+1$ collections are formed in the same way from paths not in any level h collection, $h \leq i$, and ignoring overlaps with paths in level h collections, $h \leq i$.

We will show that there are $O(n/m)$ paths in the collections containing two or more paths. We will then characterize the one path collections and show that they too contain $O(n/m)$ paths.

DEFINITION 5.9. *The* point of attachment *for a level $i+1$ collection C, $i \geq 1$, is defined in terms of C's root path ρ and the parent $\tilde{\rho}$ whose link node $\tilde{l}$ is overlapped by ρ; the point of attachment for C is link node $\tilde{l}$.*

DEFINITION 5.10. *The tree for collection C is the tree formed by the edges on its paths. The reduced tree for collection C is the subtree of its tree rooted at its point of attachment.*

LEMMA 5.11. *Each node in the tree for collection C on the path from the tree's root to its attachment point has a single child.*

Proof. Let $\tilde{C}$ be the collection containing $\tilde{\rho}$, the path whose link node is overlapped by ρ, the root path of C. Suppose, for a contradiction, that ρ' were a path in C, with start node on ρ and which branches away from ρ at or before C's point of attachment. We argue that ρ' would in fact have been added to $\tilde{C}$. Clearly, ρ''s start node lies on $\tilde{\rho}$. By assumption, ρ' does not branch away from $\tilde{\rho}$ at its link node $\tilde{l}$; hence by Lemma 5.4, it branches away before. If ρ' overlapped the link node of another path in $\tilde{C}$, then $\tilde{\rho}$ would overlap this link node too, which is not the case. Thus ρ' can be added to $\tilde{C}$, and so would not be in C, a contradiction. □

LEMMA 5.12. *Let C be a level $i+1$ collection, $i \geq 1$, ρ its root path, $\tilde{\rho}$ the path whose link node is overlapped by ρ, and $\tilde{C}$ the collection containing $\tilde{\rho}$. Then every path overlapping $\tilde{\rho}$'s link node is in C.*

Proof. By Lemma 5.5, if ρ and ρ' both overlap $\tilde{\rho}$'s link node, they do not overlap each other's link nodes. Thus they would both be added to C. □

LEMMA 5.13. *The reduced trees for the collections are disjoint.*

Proof. Without loss of generality, suppose that all the paths are connected (otherwise consider each connected component of paths separately). We will prove that not only are the reduced trees disjoint but that for each i, the reduced trees for level i collections are unrelated (for short the level i reduced trees); i.e., no tree is ancestral to another. The proof is by induction on i, the collection level. Clearly, the trees for level 1 collections are disjoint and unrelated. Now suppose that the reduced trees for each level h collection, $h \leq i$, are all disjoint and that the level i reduced trees are unrelated. Then the link nodes appearing in the level i reduced trees are also unrelated, for each link node appears on only one path in a collection. By construction, the root path of each level $i+1$ collection overlaps the link node of a path in a level i collection. By Lemma 5.12, each such root path overlaps a distinct link node, and thus the level $i+1$ reduced trees are disjoint from each other and are unrelated. □

LEMMA 5.14. *Each collection of $k > 1$ paths has its own $k-1$ distinct marked nodes.*

Proof. The marked nodes for a collection C of k paths are simply the least common ancestors (LCAs) of the link nodes for these paths. As t is binary, and these k link nodes are distinct, there are $k-1$ LCAs. Each LCA has a link node in each of its two subtrees, and consequently each LCA is marked. Finally, Lemma 5.11 implies that the LCAs all lie in the reduced tree for C, and as by Lemma 5.13 these reduced trees are disjoint, the marked nodes associated with each collection are distinct. □

COROLLARY 5.15. *There are $O(n/m)$ paths in collections of two or more paths.*

Now consider the paths in collections of size one. These paths form chains of paths in which each successive path overlaps the link node of its predecessor.

LEMMA 5.16. *There are $O(n/m)$ paths in the above chains.*

Proof. Two sets C_e and C_o of paths are formed. C_e comprises every even index path on the chains (the second, fourth, ... paths) and C_e the odd index paths.

By construction, none of the chains overlap. By Lemma 5.5, none of the paths in C_e overlap (apply the lemma to successive paths ρ_{2i}, ρ_{2i+1}, ρ_{2i+2} on a chain), and the same holds for paths in C_o.

Now we argue in a similar way to the proof of Lemma 5.11. There are $O(n/m)$ paths in C_e with one or more marked nodes (as marked nodes on distinct paths must be distinct). The remaining paths on C_e have at most one red subtree each; by property 4, each such path includes between its nodes and those of its green subtrees

at least $m/2$ distinct nodes (i.e. unshared with any other such path). This is a further $O(n/m)$ paths. Thus C_e includes $O(n/m)$ paths. The same is true for C_o. □

We have shown the following lemmas:

LEMMA 5.17. *There are $O(n/m)$ paths.*

LEMMA 5.18. *The paths have total length $O(n)$.*

This leads to the following theorem.

THEOREM 5.19. *There is a linear time reduction from the tree pattern matching problem to a collection of instances of the subset matching problem of overall linear size.*

Proof. This is immediate from Corollary 5.2 and Lemma 5.17. □

6. Further comments. It is not completely clear whether this construction maps unlabelled trees to the set strings as compactly as possible, for ancestral information is lost in the reduction. Indeed, an unlabelled n-node tree can be represented using $O(n)$ bits, whereas a size n set problem in general requires $\Theta(n \log n)$ bits and will do so after our reduction. In general, n labels would require $\Theta(n \log n)$ bits, so it appears the reduction is tight for labelled trees. Thus this raises the question of whether there are algorithms for unlabelled tree pattern matching that are faster by a $\Theta(\log n)$ factor.

Acknowledgment. We thank the referees for their suggestions which helped improve the presentation.

REFERENCES

[1] R. COLE AND R. HARIHARAN, *Tree pattern matching and subset matching in randomized $O(n \log^3 m)$ time*, in Proceedings of the 29th ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 66–75.

[2] R. COLE AND R. HARIHARAN, *Verifying candidate matches in sparse and wildcard matching*, in Proceedings of the 34th ACM Symposium on Theory of Computing, Montreal, QC, Canada, 2002, pp. 592–601.

[3] R. COLE, R, HARIHARAN, AND P. INDYK, *Tree pattern matching and subset matching in deterministic $O(n \log^3 m)$ time*, in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 1999, pp. 245–254.

[4] M. DUBINER, Z. GALIL, AND E. MAGEN, *Faster tree pattern matching*, J. ACM, 41 (1994), pp. 205–213.

[5] D. GUSFIELD, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, Cambridge, UK, 1997.

[6] C. M. HOFFMAN AND M. J. O'DONELL, *Pattern matching in trees*, J. ACM, 29 (1982), pp. 68–95.

[7] P. INDYK, *Deterministic superimposed coding with applications to pattern matching*, in Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, pp. 127–136.

[8] P. INDYK, *Faster algorithms for string matching problems: Matching the convolution bound*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1998, pp. 166–173.

[9] S. R. KOSARAJU, *Efficient tree pattern matching*, in Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, 1989, pp. 178–183.

[10] M. CROCHEMORE AND W. RYTTER, *Text Algorithms*, Oxford University Press, New York, 1994, pp. 27–31.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 1067–1072

REVISITING TUCKER'S ALGORITHM TO COLOR CIRCULAR ARC GRAPHS*

MARIO VALENCIA-PABON†

Abstract. The circular arc coloring problem consists of finding a minimum coloring of a circular arc family F such that no two intersecting arcs share a color. Let l be the minimum number of circular arcs in F that are needed to cover the circle. Tucker shows in [*SIAM J. Appl. Math.*, 29 (1975), pp. 493–502], that if $l \geq 4$, then $\lfloor \frac{3}{2}L \rfloor$ colors suffice to color F, where L denotes the load of F. We extend Tucker's result by showing that if $l \geq 5$, then $\lceil (\frac{l-1}{l-2})L \rceil$ colors suffice to color F, and this upper bound is tight.

Key words. graph algorithms, chromatic number, circular arc graphs, induced cycles

AMS subject classifications. 05C85, 05C15, 05C62, 05C38

DOI. 10.1137/S0097539700382157

1. Introduction.

The *circular arc coloring problem* consists of finding a minimum coloring of a set of arcs of a circle such that no two intersecting arcs share a color. Applications include the problem of allocating bandwidth in all-optical WDM (wavelength division multiplexing) ring networks [5, 2], compiler design, and scheduling [9].

There are several results on the circular arc coloring problem [1, 9, 4, 6, 7, 3, 5, 2]. Garey et al. [1] have shown that this problem is NP-hard, and Gavril shows in [3] that this problem remains NP-hard for Helly circular arc graphs, a subclass of circular arc graphs. Tucker [9] gave a simple 2-approximation algorithm and conjectured that $\frac{3}{2}\omega(F)$ colors are sufficient to color any family F of arcs, where $\omega(F)$ represents the size of a maximum set of pairwise intersecting arcs in F. Karapetyan [4] shows Tucker's conjecture. Recently, Kumar [5] gave a randomized $(1 + 1/e + o(1))$-approximation algorithm for instances of this problem needing at least $\omega(\ln n)$ colors, where n is the number of arcs to be colored. For special cases of this problem, Orlin, Bonuccelli, and Bovet [6] found a $O(m^2 \ln m)$ algorithm to color a family of m *proper* circular arcs (i.e., no arc is contained in any other). The result of Orlin, Bonuccelli, and Bovet was improved by Shih and Hsu [7], who found a $O(m^{1.5})$ algorithm to color a family of m proper circular arcs. Recently, Gargano and Rescigno [2] show that a family of *hereditary* circular arcs can be optimally colored in polynomial time, and they show that there exists a $\frac{4}{3}$-approximation algorithm to color a larger class of families called *quasi-hereditary* families of arcs.

In this paper we give a tighter analysis of a greedy algorithm proposed by Tucker in [9] to color circular arc graphs.

The organization of the rest of the paper is as follows. In section 2 we give the preliminaries. Section 3 contains a tighter analysis of Tucker's greedy algorithm to color circular arc graphs, and section 4 contains concluding remarks.

*Received by the editors December 8, 2000; accepted for publication (in revised form) May 6, 2003; published electronically July 8, 2003. This work was supported by the Colombian OCyT (Observatorio Colombiano de Ciencia y Tecnología) and by COLCIENCIAS under grant 0006-2001.
http://www.siam.org/journals/sicomp/32-4/38215.html

†Departamento de Matemáticas, Universidad de los Andes, Cra. 1 No. 18A - 70, Bogotá, Colombia (mvalenci@uniandes.edu.co).

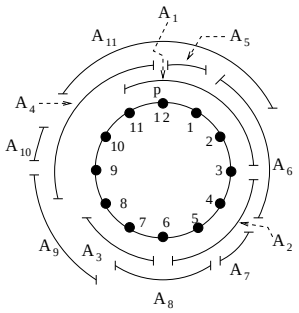
2. Preliminaries. Let $F = \{A_1, A_2, \ldots, A_m\}$ be a family of circular arcs, where each A_i represents an open real interval (a_i, b_i) on a circle, with $a_i, b_i \in \mathbb{Z}^+$ and $a_i \neq b_i$. Let n denote the largest integer among all a_i's and b_i's. Then we can regard the circle as being divided into n parts by n equally spaced points numbered clockwise as $1, 2, \ldots, n$, and each $A_i = (a_i, b_i)$ represents the circular arc from point a_i, which we call the beginning point of A_i, to point b_i, which we call the ending point of A_i, in the clockwise direction. We may assume that $n \leq 2m$ (in the worst case, the beginning and ending points of any two arcs in F are different).

Let $\chi(F)$ denote the minimum number of colors needed to color the arcs of F such that no two intersecting arcs share a color, and let $\omega(F)$ denote the size of a maximum set of pairwise intersecting arcs in F. We call the *load* of F, which we denote by $L(F)$, the largest number of arcs containing a common point on the circle. Tucker shows in [9] that $\chi(F)$ verifies $L(F) \leq \chi(F) \leq 2L(F) - 1$. Note that $L(F)$ is not necessarily equal to $\omega(F)$. In [9] some families of arcs for which $\omega(F) = 2L(F) - 1$ are given. Moreover, as $\chi(F) \geq \omega(F)$ holds for any family F of arcs, Karapetyan's result [4] is the best deterministic approximation algorithm known for this problem. We denote by $\mathcal{G}(F)$ the intersection (undirected) graph corresponding to a circular arc family F, which is called a circular arc graph. Tucker [9] gives the following greedy algorithm to properly color a family F of circular arcs.

TUCKER'S ALGORITHM.

1. Pick a point p in the circle such that S_p, the set of arcs in F containing p, has size $L(F)$. Let $A_1 = (a_1, b_1)$ be the arc in S_p that extends the least on the counterclockwise side of p, and let $cmax = 1$. Assign color $cmax$ to arc A_1.
2. For $i = 1, 2, \ldots, |F| - 1$, we move clockwise round and round the circle indexing the arc $A_{i+1} = (a_{i+1}, b_{i+1})$ in $F \setminus \{A_1, A_2, \ldots, A_i\}$ by the following rule: A_{i+1} is the first unindexed arc to begin after the ending point of A_i. Assign color $cmax$ to A_{i+1} unless there exists an arc A_j, $j < i + 1$, already colored with color $cmax$ which intersects A_{i+1}, in which case we color A_{i+1} with color $cmax + 1$ and increment $cmax$.

In Figure 2.1 we show an example of the execution of Tucker's algorithm.



Color	Arcs
1	A_1, A_2, A_3
2	A_4, A_5, A_6, A_7, A_8
3	A_9, A_{10}, A_{11}

FIG. 2.1. *Example of the execution of Tucker's algorithm.*

DEFINITION 2.1. *Let F be a family of circular arcs. We define the* circular-cover *of F to be the smallest size of any subset of arcs of F needed to cover the circle.*

Using the previous algorithm, Tucker shows the following theorem, which we rephrase in terms of the circular-cover of a family of circular arcs.

THEOREM 2.2 (see Tucker [9]). *Let F be a family of circular arcs with load*

$L = L(F)$. If the circular-cover of F is at least equal to four, then $\lfloor \frac{3}{2}L \rfloor$ colors suffice to color F.

Note that it must often be the case in practical situations that the circular-cover of a family of arcs is significantly greater than four. For instance, consider the task of assigning workers to jobs in a cyclic schedule. The time periods occupied by the jobs are the arcs, the colors assigned to the arcs are the workers, and the constraint that no worker can work on two jobs at once is realized by the constraint that no intersecting arcs can have the same color. The goal is to minimize the number of workers employed. Now, if the durations of the jobs are short compared with the period of the cyclic schedule, then the circular-cover of the corresponding family of arcs will be large. Therefore, in the following section, we analyze the performance of Tucker's greedy algorithm to color circular arc graphs in terms of the circular-cover of the associated family of arcs.

3. Analysis of Tucker's greedy algorithm. In this section we give a detailed analysis of Tucker's algorithm. Our main result is the following theorem, which extends Tucker's given in Theorem 2.2.

THEOREM 3.1. *Let F be a family of circular arcs with load $L = L(F)$. If the circular-cover of F is equal to l, with $l \geq 5$, then $\lceil (\frac{l-1}{l-2})L \rceil$ colors suffice to color F.*

In order to prove Theorem 3.1, we need the following definitions. Let F be a family of circular arcs with load $L(F)$, and assume that we use Tucker's greedy algorithm to color the arcs in F. Let $A_1 = (a_1, b_1)$ be the first arc in F considered by Tucker's algorithm. Let us denote the beginning point of arc A_1 as t. We say that Tucker's algorithm has completed k rounds if the algorithm has traversed the point t $k+1$ times, where arc A_1 counts. We let T_k be the set of arcs colored during the kth round, and we let $F_k = \bigcup_{i=1}^{k} T_i$ be the subfamily of arcs colored until the kth round.

Proof of Theorem 3.1. Let m be the largest integer used in the description of the arcs in F. W.l.o.g. assume that each two consecutive points p and $p+1$ on the circle, $1 \leq p \leq m$, are traversed by exactly L arcs from F. Otherwise, if some two consecutive points p and $p+1$ are traversed by $r < L$ arcs, then we add $L - r$ arcs of the form $(p, p+1)$ (or $(m, 1)$ if $p = m$) to F without changing its colorability. We assume that we use Tucker's algorithm to color the arcs in F, and we proceed by induction on L. If $L = 1$, the result is trivial. Assume that the result holds when $L(F) < L$ for any F and some $L \geq 2$, and we shall prove that the result holds for any F with $L(F) = L$. By hypothesis, the circular-cover l of F verifies $l \geq 5$. Let A_1 be the first arc in F colored by Tucker's algorithm, and let t be the beginning point of arc A_1. For $i = 1, 2, \ldots$, let A_i denote the first arc in F colored i. Thus, considering the way Tucker's algorithm colors the arcs in F, we can deduce the following properties. We prove only Property 3, because Properties 1 and 2 can be easily deduced.

PROPERTY 1. *For each i, $i > 1$, A_i intersects A_{i-1}.*

PROPERTY 2. *For each i, $i \geq 1$, $L(F \setminus F_i) \leq L(F) - i$, where $L(F \setminus F_i)$ is the load induced by the subfamily of arcs $F \setminus F_i$.*

PROPERTY 3. *For each i, $1 \leq i \leq l-2$, Tucker's algorithm uses at most $i+1$ colors to color the arcs in F_i, where l is the circular-cover of the family F.*

Proof of Property 3. We proceed by induction on i. If $i = 1$, the result is trivial. Assume that the result holds for $j < i$, and we will prove that the result holds for $j = i$. Let $i > 1$. Let A_i be the first arc in T_{i-1} colored with color i, and let A_i^f be the last arc in T_{i-1} colored with color i that traverses the point t for the ith time (note that A_i and A_i^f can be the same arc (see Figure 3.1(b)). By induction hypothesis, these arcs exist; otherwise, the number of colors used by the algorithm to color the

arcs in F_{i-1} is at most $i-1$, and the result trivially follows. Now, let A_{i+1} be the first arc in T_i colored with color $i+1$. By Property 1, we have that A_{i+1} intersects A_i. Let A^f_{i+1} be the last arc in T_i that traverses the point t for the $(i+1)$th time. Suppose that A^f_{i+1} cannot be colored with color $i+1$. Thus, by Property 1, we have that A_{i+1} intersects A^f_{i+1}, and so the $i+1$ arcs $A_2, A_3, \ldots, A_i, A_{i+1}, A^f_{i+1}$ cover the circle (see Figure 3.1(c)). However, $i \leq l-2$, which implies that the circular-cover of F is at most $l-1$, which is a contradiction. This ends the proof of this property.

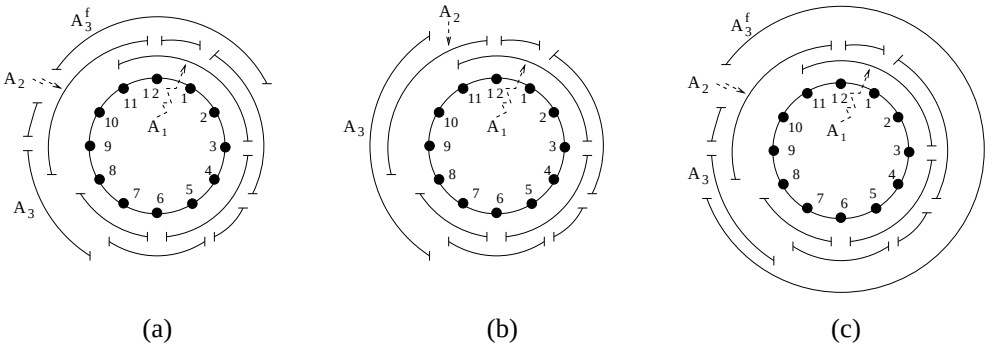


FIG. 3.1. (a), (b) *Examples of possible configurations in the proof of Property* 3 *for* $i = 2$ *and* $l = 7$. (c) *Example of forbidden configuration in the proof of Property* 3 *for* $i = 2$ *and* $l = 7$.

Now, the proof of Theorem 3.1 can be easily deduced from Properties 2 and 3 as follows. By Property 2, we have that $L(F \setminus F_{l-2}) \leq L-(l-2)$ and, by Property 3, we have that Tucker's algorithm uses at most $l-1$ colors to color the arcs in F_{l-2}. Therefore, by induction hypothesis, the theorem holds. □

The following corollary can be deduced directly from Theorem 3.1.

COROLLARY 3.2. *Let F be a family of circular arcs with load $L = L(F)$ on a circle on n points. For all $k \geq 3$, if each arc in F spans at most n/k points of the circle, then the number of colors used by Tucker's algorithm to color F is at most* $\lceil (\frac{k}{k-1})L \rceil$.

It is easy to verify that in the complementary case of Corollary 3.2, i.e., when each circular arc in a family F spans more than half of the points in a circle, the intersection graph of the arcs in F is a complete graph on $|F|$ vertices.

PROPOSITION 3.3. *Let F be a family of circular arcs on a circle. If each arc in F spans more than half of the points in the circle, then F can be optimally colored in linear time.*

The following proposition gives us an idea of the tightness of the upper bound obtained in Theorem 3.1.

PROPOSITION 3.4. *For any odd integer l, $l \geq 3$, and for any even integer L, $L \geq 2$, there exists a family F of circular arcs with load L and circular-cover l such that* $\chi(F) = \lceil (\frac{l}{l-1})L \rceil$.

Proof. In order to prove Proposition 3.4, we use a result of Stahl [8] concerning the *r-tuple* colorings of graphs. An r-tuple coloring of a graph G is an assignment of r distinct colors to each vertex of G in such a manner that no two adjacent vertices share a color. In the special case of a cycle C_l with odd length l, $l \geq 3$, Stahl shows in [8] that the minimum number of colors for an r-tuple coloring of C_l, $r \geq 1$, is equal to $2r+1+\lfloor \frac{2(r-1)}{l-1} \rfloor$. Thus, F is constructed as follows. Let $r = L/2$, and assume that

the vertices of C_l are labeled clockwise by $0, 1, \ldots, l-1$. For all $i = 0, 1, \ldots, l-1$, F contains r copies of the arc $(i, i+2 \mod l)$. By construction, it is easy to verify that if $L = 2$ (and thus $r = 1$), then the intersection graph of the arcs in F is isomorphic to C_l and therefore the circular-cover of F is equal to l. Hence, for any even integer $L = 2r$, with $r \geq 1$, there exists a family F of circular arcs with load L and circular-cover l such that $\chi(F) = 2r + 1 + \lfloor \frac{2r-2}{l-1} \rfloor = L + 1 + \lfloor \frac{L-2}{l-1} \rfloor = \lceil (\frac{l}{l-1})L \rceil$. □

Note that we cannot hope that, for any family F with load L and circular-cover $l \geq 5$, $\lfloor (\frac{l-1}{l-2})L \rfloor$ colors suffice to color the arcs in F. In fact, by Proposition 3.4, we have that any cycle graph C_l with odd length l, $l \geq 5$, can be represented by a family of l arcs with load 2 and circular-cover equal to l. Thus, $\lfloor (\frac{l-1}{l-2})2 \rfloor = 2$, but the chromatic number of C_l is 3. The following proposition shows that there are instances of the circular arc coloring problem where the upper bound in Theorem 3.1 is reached.

PROPOSITION 3.5. *The upper bound in Theorem* 3.1 *is tight.*

Proof. Let C_l be a cycle graph on l vertices, with $l = 2k + 1$ for any positive integer $k \geq 2$. By Proposition 3.4, we have that for all r, $1 \leq r \leq k-1$, an r-tuple coloring of C_l can be represented by a family F of arcs with load $L = 2r$ and circular-cover l. Moreover, the minimum number of colors needed to color F is exactly $2r + 1$. By Theorem 3.1, we have that the arcs in F can be colored using at most $\lceil (\frac{l-1}{l-2})L \rceil = \lceil \frac{4kr}{2k-1} \rceil = \lceil 2r + \frac{2r}{2k-1} \rceil = 2r + 1$ colors. This ends the proof of this proposition. □

Let F be a family of arcs with load $L(F)$. By Property 2 in Theorem 3.1, we have that for each i, $i \geq 1$, the load of the subfamily $F \setminus F_i$ is at most equal to $L(F) - i$. Moreover, for each round i, Tucker's algorithm uses at most two new colors to color the arcs in T_i. Therefore, in the general case (i.e., if the circular-cover l of F verifies $l < 4$), the number of colors used by Tucker's algorithm to color the arcs in F is at most equal to $2L(F)$, and thus it gives a 2-approximation algorithm for the circular arc coloring problem, as noted earlier by Tucker in [9].

4. Conclusion. We have shown a tighter analysis of Tucker's greedy algorithm to color a family of circular arcs. We have proved that Tucker's algorithm is quasi-optimal for families of arcs having a large circular-cover. Our results extend a previous one given by Tucker. An interesting open problem is the complexity of the circular arc coloring problem for families of arcs having bounded spans and unbounded load.

Acknowledgment. The author gratefully acknowledges the helpful comments and suggestions of two anonymous referees.

REFERENCES

[1] M. R. GAREY, D. S. JOHNSON, G. L. MILLER, AND C. H. PAPADIMITRIOU, *The complexity of coloring circular arcs and chords*, SIAM J. Alg. Disc. Meth., 1 (1980), pp. 216–227.

[2] L. GARGANO AND A. RESCIGNO, *Coloring circular arcs with applications to WDM routing*, in Proceedings of the Workshop on Approximation and Randomization Algorithms in Communication Networks, Geneva, Switzerland, 2000.

[3] F. GAVRIL, *Intersection graphs of Helly families of subtrees*, Discrete Appl. Math., 66 (1996), pp. 45–56.

[4] I. A. KARAPETYAN, *Coloring of arc graphs*, Akad. Nauk Armyan. SSR Dokl., 70 (1980), pp. 306–311 (in Russian).

[5] V. KUMAR, *An approximation algorithm for circular arc coloring*, Algorithmica, 30 (2001), pp. 406–417.

[6] J. B. Orlin, M. A. Bonuccelli, and D. P. Bovet, *An $O(n^2)$ algorithm for coloring proper circular arc graphs*, SIAM J. Alg. Disc. Meth., 2 (1981), pp. 88–93.
[7] W.-K. Shih and W.-L. Hsu, *An $O(n^{1.5})$ algorithm to color proper circular arcs*, Discrete Appl. Math., 25 (1989), pp. 321–323.
[8] S. Stahl, *n-tuple colorings and associated graphs*, J. Combinatorial Theory Ser. B, 20 (1976), pp. 185–203.
[9] A. Tucker, *Coloring a family of circular arcs*, SIAM J. Appl. Math., 29 (1975), pp. 493–502.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 1073–1090

GENETIC DESIGN OF DRUGS WITHOUT SIDE-EFFECTS*

XIAOTIE DENG†, GUOJUN LI‡, ZIMAO LI§, BIN MA¶, AND LUSHENG WANG†

Abstract. Consider two sets of strings, $\mathcal{B}$ (bad genes) and $\mathcal{G}$ (good genes), as well as two integers d_b and d_g ($d_b \leq d_g$). A frequently occurring problem in computational biology (and other fields) is to find a (distinguishing) substring s of length L that distinguishes the bad strings from good strings, i.e., such that for each string $s_i \in \mathcal{B}$ there exists a length-L substring t_i of s_i with $d(s, t_i) \leq d_b$ (close to bad strings), and for every substring u_i of length L of every string $g_i \in \mathcal{G}$, $d(s, u_i) \geq d_g$ (far from good strings).

We present a polynomial time approximation scheme to settle the problem; i.e., for any constant $\epsilon > 0$, the algorithm finds a string s of length L such that for every $s_i \in \mathcal{B}$ there is a length-L substring t_i of s_i with $d(t_i, s) \leq (1 + \epsilon)d_b$, and for every substring u_i of length L of every $g_i \in \mathcal{G}$, $d(u_i, s) \geq (1 - \epsilon)d_g$ if a solution to the original pair ($d_b \leq d_g$) exists. Since there is a polynomial number of such pairs (d_b, d_g), we can exhaust all the possibilities in polynomial time to find a good approximation required by the corresponding application problems.

Key words. approximation algorithms, computational molecular biology, distinguishing substring selection

AMS subject classifications. 68W25, 90C27, 92D20

DOI. 10.1137/S0097539701397825

1. Introduction.

Research effort in molecular biology, such as the human genome project, has been revealing the secret of our genetic composition, the long DNA sequences that can determine many aspects of life. Applications that use this information have posed new challenges to the design and analysis of efficient computational methods.

A frequently recurring problem in biological applications is to find one substring of length L that appears (with a few substitutions) at least once in each of a set of bad strings (such as bacterial sequences) and is not "close" to any substring of length L in *each* of another set of good strings (such as human and livestock sequences). The problem has various applications in molecular biology such as the design of universal PCR primers, identification of genetic drug targets, and design of genetic probes [8, 2, 12, 5]. In particular, the genetic drug target identification problem searches for a sequence of genes that is close to bad genes (the target) but far from all good genes (to avoid side-effects). Our study develops a polynomial time approximation scheme in both measures simultaneously.

*Received by the editors November 12, 2001; accepted for publication (in revised form) April 18, 2003; published electronically July 11, 2003.

http://www.siam.org/journals/sicomp/32-4/39782.html

†Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong (deng@cs.cityu.edu.hk, lwang@cs.cityu.edu.hk). The first author was supported by a grant from the National Natural Science Foundation of China and Research Grants Council of the HKSAR Joint Research Scheme (project N_CityU 102/01). The last author was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (project CityU 1047/01E).

‡School of Mathematics and System Sciences, Shandong University, Jinan 250100, P. R. China, and Institute of Software, Chinese Academy of Science, Beijing 100080, P. R. China (gjli@sdu.edu.cn).

§School of Computer Science and Technology, Shandong University, Jinan 250100, P. R. China (lizm@sdu.edu.cn). This author was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (project CityU 1047/01E).

¶Department of Computer Science, University of Western Ontario, London, ON N6A 5B7, Canada (bma@csd.uwo.ca).

1.1. The mathematical model. The *distinguishing substring selection* problem has as input two sets of strings, $\mathcal{B}$ and $\mathcal{G}$. It is required to find a substring of unspecified length (denoted by L) that is, informally, close to a substring of every string in $\mathcal{B}$ and far away from every length-L substring of strings in $\mathcal{G}$. Since $\mathcal{G}$ can be reconstructed to contain all substrings of length L in each of all the good strings, we can assume that every string in $\mathcal{G}$ has the same length L.

Therefore, here we formally define the problem as follows: Given a set $\mathcal{B} = \{s_1, s_2, \ldots, s_{n_1}\}$ of n_1 (bad) strings of length at least L, a set $\mathcal{G} = \{g_1, g_2, \ldots, g_{n_2}\}$ of n_2 (good) strings of length exactly L, and two integers d_b and d_g ($d_b \leq d_g$), the distinguishing substring selection problem (DSSP) is to find a string s such that for each string $s_i \in \mathcal{B}$ there exists a length-L substring t_i of s_i with $d(s, t_i) \leq d_b$, and for any string $g_i \in \mathcal{G}$, $d(s, g_i) \geq d_g$. Here $d(,)$ represents the Hamming distance between two strings. If all strings in $\mathcal{B}$ are also of the same length L, the problem is called the distinguishing string problem (DSP).

1.2. Previous results. The best previous known approximation ratio for DSSP (and DSP) is 2 [5]. In another direction, many simplified versions were proposed to make the task easier in terms of computation/approximation, conceding more general applicability.

The problem is NP-hard, even when only one objective is to be met [1, 5]. To approximate the center string that is far away from each of the good strings is not difficult and is shown to have a polynomial time approximation scheme by Lanctot et al. [5]. The problem of finding a center string s that is close to all the bad strings is nontrivial, and it has taken intensive investigation by several research groups [5, 3] to finally develop a polynomial time approximation scheme [6].

There are also several modifications and variations on this initial strategy, e.g., the *d-characteristic string* [4], the *far from most string* [5], and the formulation of Ben-Dor et al. [1].

The DSSP problem that contains two objective functions has remained elusive despite the above-mentioned intensive research effort. To meet the requirements of many application problems, however, we would need to have a solution for DSSP.

1.3. Our contribution. In this paper, we settle this difficult problem by presenting a polynomial time approximation scheme in both requirements. If a solution to the original problem exists, for any constant $\epsilon > 0$, our algorithm computes a string s of length L such that for every $s_i \in \mathcal{S}$ there is a length-L substring t_i of s_i with $d(t_i, s) \leq (1 + \epsilon)d_b$, and for every $g_i \in \mathcal{G}$, $d(g_i, s) \geq (1 - \epsilon)d_g$. Here d_b and d_g are two important parameters supplied by users to specify the distances from the distinguishing string s to the set of bad strings and the set of good strings. Our algorithm gives a solution that approximately meets the requirements, i.e., $(1 + \epsilon)d_b$ and $(1 - \epsilon)d_g$, if a solution to the original requirements, i.e., d_b and d_g, exists.

1.4. Sketch of our approach. Design and analysis of good algorithms for approximating multiple objective functions are not simple in general (see [11] for a general approach and related works).

The standard techniques for related center string problems follow a linear program approach combined with randomized rounding. That works for DSP and DSSP when the parameters are sufficiently large. The main difficulty for our problem lies in the case when objective function value is relatively small but still too large for enumeration methods to work.

To overcome this difficulty, we sample a constant number of strings and find the

positions at which the samples have the same letters (bases). We denote this set of positions by Q and the set of the rest by P. Our breakthrough starts with a key lemma that states that there is a set of samples for which there are y positions in Q such that, when we choose the y positions in Q carefully, choose the letters of the rest of $|Q|-y$ positions to be the same as the samples, and choose letters at positions in P by the linear program approach, then we can obtain the right approximate solution.

An interesting case is when $y < O(\log(n))$, i.e., y is small, but not small enough for a brute-force enumeration method to go through directly. A new method is designed to handle this case. Since similar situations occur in many combinatorial optimization problems, we expect that this idea may have wider application.

1.5. Organization of the paper. We focus on the polynomial time approximation scheme for DSP in sections 2 through 4. Section 2 introduces the related notation and the standard integer programming formulation, which works well when both d_b and d_g are large, i.e., at least $\Omega(L)$. Section 3 gives the key lemma. We discuss the methods for finding a good approximation of the set of y positions in Q in section 4.

In section 5, we show that the established algorithm for DSP can be extended to work for the general case, the DSSP. We conclude our work in section 6 with discussion and remarks.

2. Preliminaries. We consider two sets of strings: $\mathcal{G}$ (good strings) and $\mathcal{B}$ (bad strings). We call d_b the *upper* radius for bad strings ($\mathcal{B}$) and d_g the *lower* radius for good strings ($\mathcal{G}$). Let $n = n_1 + n_2$ be the total number of good and bad strings in $\mathcal{G} \cup \mathcal{B}$.

For the DSP, every good or bad string is of the same length L. The distance $d(x, y)$ of two strings x and y is their Hamming distance, i.e., the number of positions at which they differ from each other. We are to find the string x of length L such that

$$\begin{cases} d(s_i, x) \le d_b, \quad s_i \in \mathcal{B}, & i = 1, \ldots, n_1, \\ d(g_j, x) \ge d_g, \quad g_j \in \mathcal{G}, & j = 1, \ldots, n_2. \end{cases} \tag{1}$$

In this section, we present an approximation algorithm that works well for a special case of the DSP. The restriction is that L, d_b, and d_g are large; more specifically, $L \ge (4\log(n_1 + n_2))/\epsilon^2$, $d_g = \Omega(L)$, and $d_b = \Omega(L)$, where ϵ is the parameter controlling the performance ratio and n_1 and n_2 are the numbers of bad and good strings, respectively.

This is achieved via a standard method using integer linear programming. Define $\chi(s, i, k, a) = 0$ if $s_i[k] = a$, and $\chi(s, i, k, a) = 1$ if $s_i[k] \ne a$. Similarly, define $\chi(g, j, k, a) = 0$ if $g_j[k] = a$, and $\chi(g, j, k, a) = 1$ if $g_j[k] \ne a$. The problem becomes the following integer linear programming (LP) problem:

$$\begin{cases} \sum_{a \in \Sigma} x_{k,a} = 1, \quad k = 1, 2, \ldots, L, \\ \sum_{1 \le k \le L} \sum_{a \in \Sigma} \chi(s, i, k, a)\, x_{k,a} \le d_b, \quad i = 1, 2, \ldots, n_1, \\ \sum_{1 \le k \le L} \sum_{a \in \Sigma} \chi(g, j, k, a)\, x_{k,a} \ge d_g, \quad j = 1, 2, \ldots, n_2, \\ x_{k,a} \in \{0, 1\}, \quad a \in \Sigma, \ k = 1, 2, \ldots, L. \end{cases} \tag{2}$$

Let $\bar{x}_{k,a}$ be a solution for LP relaxation of (2). For each $0 \le k \le L$, with probability $\bar{x}_{k,a}$, we set $x_{k,a} = 1$ and set $x_{k,a'} = 0$ for any $a' \ne a$. We choose the random variables independently for different k. This results in an integer solution for (2) (and hence a solution for (1)) if d_b is replaced by $(1 + \epsilon)d_b$ and d_g by $(1 - \epsilon)d_g$, as shown in Theorem 2. Standard derandomization methods [10] transfer it to a

deterministic algorithm in Theorem 3. The following lemma is useful for the proof of Theorem 2.

LEMMA 1. *Let $X_1, X_2, \ldots, X_n$ be n independent random* 0-1 *variables, where X_i takes* 1 *with probability p_i, $0 < p_i < 1$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$. Then for any $0 < \delta \leq 1$, $\mathbf{Pr}(X > \mu + \delta n) < \exp(-\frac{1}{3}n\delta^2)$, and $\mathbf{Pr}(X < \mu - \delta n) \leq \exp(-\frac{1}{2}n\delta^2)$.*

THEOREM 2. *Let $\delta > 0$ be any constant. Suppose that $L \geq (4\log(n_1 + n_2))/\delta^2$, $d_g \geq c_g L$, and $d_b \geq c_b L$. There is a randomized algorithm that finds a solution, i.e., a string x of length L, with high probability such that for each string s_i in $\mathcal{B}$, $d(s_i, x) < (1 + \delta/c_b)d_b$, and for any string g_j in $\mathcal{G}$, $d(g_j, x) > (1 - \delta/c_g)d_g$.*

Proof. We can see that $\sum_{a\in\Sigma} \chi(s, i, k, a)\, x_{k,a}$ and $\sum_{a\in\Sigma} \chi(g, j, k, a)\, x_{k,a}$ take 1 or 0 randomly and independently for different k's. Thus we have that $d(s, i, x) = \sum_{1\leq k\leq L} \sum_{a\in\Sigma} \chi(s, i, k, a)\, x_{k,a}$ is a sum of L independent 0-1 random variables. Similarly, $d(g, j, x) = \sum_{1\leq k\leq L} \sum_{a\in\Sigma} \chi(g, j, k, a)\, x_{k,a}$ is also a sum of L independent 0-1 random variables. Moreover,

$$\begin{aligned} E[d(s, i, x)] &= \sum_{1\leq k\leq L} \sum_{a\in\Sigma} \chi(s, i, k, a)\, E[x_{k,a}] \\ &= \sum_{1\leq k\leq L} \sum_{a\in\Sigma} \chi(s, i, k, a)\, \bar{x}_{k,a} \\ &\leq \bar{d} \leq d_b, \end{aligned} \tag{3}$$

$$\begin{aligned} E[d(g, j, x)] &= \sum_{1\leq k\leq L} \sum_{a\in\Sigma} \chi(g, j, k, a)\, E[x_{k,a}] \\ &= \sum_{1\leq k\leq L} \sum_{a\in\Sigma} \chi(g, j, k, a)\, \bar{x}_{k,a} \\ &\geq d_g. \end{aligned} \tag{4}$$

Thus, for any fixed $\delta > 0$, using Lemma 1, we have

$$\mathbf{Pr}\left(d(s, i, x) > d_b + \delta L\right) < \exp\left(-\frac{1}{3}\delta^2 L\right),$$

$$\mathbf{Pr}\left(d(g, j, x) < d_g - \delta L\right) \leq \exp\left(-\frac{1}{2}\delta^2 L\right).$$

Considering all bad and good strings, respectively, we have

$$\mathbf{Pr}\left(d(s, i, x) > d_b + \delta L \text{ for at least one } s_i \in \mathcal{B}\right) < n_1 \times \exp\left(-\frac{1}{3}\delta^2 L\right)$$

and

$$\mathbf{Pr}\left(d(g, j, x) < d_g - \delta L \text{ for at least one } g_j \in \mathcal{G}\right) \leq n_2 \times \exp\left(-\frac{1}{2}\delta^2 L\right).$$

When $L \geq (4\log(n_1 + n_2))/\delta^2$, we get $n_1 \times \exp(-\frac{1}{3}\delta^2 L) \leq n_1^{-\frac{1}{3}}$ and $n_2 \times \exp(-\frac{1}{2}\delta^2 L) \leq n_2^{-1}$. For $d_b = \Omega(L)$ and $d_g = \Omega(L)$, there exist positive constants c_b and c_g such that $d_b \geq c_b L$ and $d_g \geq c_g L$. Thus we get a randomized algorithm to find a

solution x for (1) with probability at least $1-(n_1^{-\frac{1}{3}}+n_2^{-1})$ such that for $i=1,2,\ldots,n_1$, $d(s,i,x) \leq (1+\delta/c_b)d_b$, and for $j=1,2,\ldots,n_2$, $d(g,j,x) \geq (1-\delta/c_g)d_g$. □

THEOREM 3. *There exists a polynomial time approximation scheme (PTAS) for a DSP when $d_g = \Omega(L)$ and $d_b = \Omega(L)$. Specifically, when $d_g = c_g \times L$ and $d_b = c_b \times L$, the PTAS runs in $O(L \times |\Sigma|^{(4\log(n_1+n_2))/\delta^2})$ time and finds a solution, i.e., a string x of length L, such that for each string s_i in $\mathcal{B}$, $d(s_i,x) < (1+\epsilon)d_b$, and for any string g_j in $\mathcal{G}$, $d(g_j,x) > (1-\epsilon)d_g$, where $\delta = \epsilon \times \min\{c_g, c_b\}$.*

Proof. When $L \geq (4\log(n_1+n_2))/\delta^2$, the following derandomized algorithm is a PTAS.

Let $x_{j,a}$ be a fractional solution for (2) for $j=1,2,\ldots,L$. We can arbitrarily decompose the L positions into disjoint sets of positions $L_1, L_2, \ldots, L_k$ such that $|L_i| \geq (4\log(n_1+n_2))/\delta^2$ and $|L_i| < (8\log(n_1+n_2))/\delta^2$ for $i=1,2,\ldots,k$. Let $\mu_{s,l,i} = \sum_{1\leq j\leq |L_i|}\sum_{a\in\Sigma}\chi(s,l,j,a)x_{j,a}$ and $\mu_{g,l,i} = \sum_{1\leq j\leq |L_i|}\sum_{a\in\Sigma}\chi(g,l,j,a)x_{j,a}$. For each L_i, replacing $\{1,2,\ldots,L\}$ with L_i in (2), we know that there exists a string x_i of length $|L_i|$ such that for each string $s_l \in \mathcal{B}$,

$$d(s_l|_{L_i}, x_i) \leq \mu_{s,l,i} + \delta|L_i|,$$

and for each string $g_l \in \mathcal{G}$,

$$d(g_l|_{L_i}, x_i) \geq \mu_{g,l,i} - \delta|L_i|.$$

Thus, we can enumerate all strings of length $|L_i|$ in $|\Sigma|^{(4\log(n_1+n_2))/\delta^2}$ time to determine x_i. Concatenating all x_i's, we have a string x of length L such that for each string $s_l \in \mathcal{B}$,

$$d(s_l,x) \leq \sum_{i=1}^{k}(\mu_{s,l,i} + \delta|L_i|) = d_b + \delta L \leq d_b + \epsilon d_b,$$

and for each string $g_l \in \mathcal{G}$,

$$d(g_l,x) \geq \sum_{i=1}^{k}(\mu_{l,i} - \delta|L_i|) = d_g - \delta L \geq d_g - \epsilon d_g.$$

Thus, we obtain a desired string x in $O(L \times |\Sigma|^{(4\log(n_1+n_2))/\delta^2})$ time, which is polynomial in terms of the input size when $|\Sigma|$ is a constant.

When $L \leq (4\log(n_1+n_2))/\delta^2$, we can enumerate all possible strings of length L in $O(|\Sigma|^{(4\log(n_1+n_2))/\delta^2})$ time. Thus, we can get a desired solution. □

3. A key lemma. To obtain our PTAS algorithm, we need to introduce two parameters: an integer r and a positive number δ. The constant r is introduced in this section and depends purely on ϵ. The constant $\delta > 0$ will be introduced in the next section and depends on both $\epsilon > 0$ and r.

For any $r > 2$, let $1 \leq i_1, i_2, \ldots, i_r \leq n_1$ be r distinct numbers. Let $Q_{i_1,i_2,\ldots,i_r}$ be the set of positions at which $s_{i_1}, s_{i_2}, \ldots, s_{i_r}$ agree. Let s be a feasible solution of length L for the distinguishing string selection problem with the upper radius d_b for bad strings ($\mathcal{B}$) and lower radius d_g for good strings ($\mathcal{G}$). $P_{i_1,i_2,\ldots,i_r} = \{1,2,\ldots,L\} - Q_{i_1,i_2,\ldots,i_r}$ is the set of positions at which at least one pair of strings s_{i_j} and $s_{i_{j'}}$ differ. Then, at such positions, s differs from at least one of s_{i_j}. The number of positions at which s differs from one of s_{i_j} is no more than d_b. Therefore, the total number of

positions at which s differs from at least one of s_{i_j} $(j = 1, 2, \ldots, r)$ is no more than rd_b. It follows that $rd_b \geq |P_{i_1,i_2,\ldots,i_r}| = L - |Q_{i_1,i_2,\ldots,i_r}|$. That is, $|Q_{i_1,i_2,\ldots,i_r}| \geq L - rd_b$. In addition, we immediately obtain the following result, which is often applied late in the proof of our main result.

CLAIM 4. *Assuming that $d_g \geq d_b$, we have $rd_g \geq rd_b \geq L - |Q_{i_1,i_2,\ldots,i_r}| = |P_{i_1,i_2,\ldots,i_r}|$.*

Let $\mathcal{B} = \{s_1, s_2, \ldots, s_{n_1}\}$. Let $p_{i_1,i_2,\ldots,i_k}$ be the number of mismatches between s_{i_1} and s at the positions in $Q_{i_1,i_2,\ldots,i_k}$. Let $\rho_k = \min_{1 \leq i_1,i_2,\ldots,i_k \leq n_1} p_{i_1,i_2,\ldots,i_k}/d_b$. The following claim is a variant of a claim in [6], but the proof is identical.

CLAIM 5. *For any k such that $2 \leq k \leq r$, where r is a constant, there are indices $1 \leq i_1, i_2, \ldots, i_r \leq n_1$ such that for any $1 \leq l \leq n_1$*

$$|\{j \in Q_{i_1,i_2,\ldots,i_r} \mid s_{i_1}[j] \neq s_l[j] \text{ and } s_{i_1}[j] \neq s[j]\}| \leq (\rho_k - \rho_{k+1})\, d_b.$$

Proof. Consider indices $1 \leq i_1, i_2, \ldots, i_k \leq n$ such that $p_{i_1,i_2,\ldots,i_k} = \rho_k d_{opt}$. Then for any $1 \leq i_{k+1}, i_{k+2}, \ldots, i_r \leq n$ and $1 \leq l \leq n$ we have

$$\begin{aligned}
&|\{j \in Q_{i_1,i_2,\ldots,i_r} \mid s_{i_1}[j] \neq s_l[j] \text{ and } s_{i_1}[j] \neq s[j]\}| \\
&\leq |\{j \in Q_{i_1,i_2,\ldots,i_k} \mid s_{i_1}[j] \neq s_l[j] \text{ and } s_{i_1}[j] \neq s[j]\}| \qquad (5)\\
&= |\{j \in Q_{i_1,i_2,\ldots,i_k} \mid s_{i_1}[j] \neq s[j]\} \\
&\quad - \{j \in Q_{i_1,i_2,\ldots,i_k} \mid s_{i_1}[j] = s_l[j] \text{ and } s_{i_1}[j] \neq s[j]\}| \\
&= |\{j \in Q_{i_1,i_2,\ldots,i_k} \mid s_{i_1}[j] \neq s[j]\} - \{j \in Q_{i_1,i_2,\ldots,i_k,l} \mid s_{i_1}[j] \neq s[j]\}| \\
&= p_{i_1,i_2,\ldots,i_k} - p_{i_1,i_2,\ldots,i_k,l} \qquad (6)\\
&\leq (\rho_k - \rho_{k+1})\, d_{opt},
\end{aligned}$$

where inequality (5) holds because $Q_{i_1,i_2,\ldots,i_r} \subseteq Q_{i_1,i_2,\ldots,i_k}$ and equality (6) holds because $Q_{i_1,i_2,\ldots,i_k,l} \subseteq Q_{i_1,i_2,\ldots,i_k}$. ∎

From Claim 5, we can immediately obtain the following proposition.

PROPOSITION 6. *For any constant r, there are indices $1 \leq i_1, i_2, \ldots, i_r \leq n_1$ such that for any $s_l \in \mathcal{B}$,*

$$|\{j \in Q_{i_1,i_2,\ldots,i_r} \mid s_{i_1}[j] \neq s_l[j] \text{ and } s_{i_1}[j] \neq s[j]\}| \leq \frac{1}{r-1}\, d_b.$$

Proof. Note that

$$(\rho_2 - \rho_3) + (\rho_3 - \rho_4) + \cdots + (\rho_r - \rho_{r+1}) = \rho_2 - \rho_{r+1} \leq 1.$$

Therefore, one of $(\rho_k - \rho_{k+1})$ is at most $\frac{1}{r-1}$. Thus, Claim 5 immediately implies that the lemma is true. ∎

The following corollary will also be useful later.

COROLLARY 7. *For any $Q \subseteq Q_{i_1,i_2,\ldots,i_r}$ (as in Proposition 6), we have that, for any $s_l \in \mathcal{B}$,*

$$|\{j \in Q \mid s_{i_1}[j] \neq s_l[j] \text{ and } s_{i_1}[j] \neq s[j]\}| \leq \frac{1}{r-1}\, d_b.$$

Moreover, let $P = \{1, 2, \ldots, L\} - Q$. Then, for any $s_l \in \mathcal{B}$,

$$d(s_l, s_{i_1}|Q) + d(s_l, s|P) \leq \frac{r}{r-1} d_b.$$

Proof. The first part is obvious by Proposition 6. For the second part, consider

$$A_k = \{j \in Q \mid s_{i_1}[j] \neq s_k[j] \text{ and } s_{i_1}[j] \neq s[j]\}$$

and

$$B_k = \{j \in Q \mid s_{i_1}[j] \neq s_k[j] \text{ and } s_{i_1}[j] = s[j]\}.$$

Then $d(s_k, s_{i_1}|Q) = |A_k| + |B_k|$. By the first part of the corollary, $|A_k|$ is no more than $\frac{1}{r-1}d_b$. By definition, $|B_k| \leq d(s_k, s|Q)$. Therefore, $|B_k| + d(s_k, s|P) \leq d(s_k, s|Q) + d(s_k, s|P) = d(s_k, s) \leq d_b$. The rest of the corollary follows. □

Informally, Proposition 6 implies that s_{i_1} is a good approximation of s at positions in $Q_{i_1,i_2,\ldots,i_r}$ for the bad strings; i.e., for any $Q \subseteq Q_{i_1,i_2,\ldots,i_r}$,

$$d(s_l, s_{i_1}|Q) \leq d(s_l, s|Q) + \frac{1}{r-1}\, d_b.$$

Before we present our key lemma, we need a boosting proposition that, when applied together with Corollary 7, obtains a better and better solution.

PROPOSITION 8. *Let $Q \subset Q_{i_1,i_2,\ldots,i_r}$ (as in Proposition 6). Consider the index $k : 1 \leq k \leq n_1$ and the number $y \geq 0$ such that, for any $s_l \in \mathcal{B}$, $d(s_l, s_{i_1}|Q) + d(s_l, s|P) \leq d(s_k, s_{i_1}|Q) + d(s_k, s|P) = \frac{r}{r-1}d_b - y$ (where $P = \{1, 2, \ldots, L\} - Q$). Then we have*

$$|\{j \in Q^k | s_{i_1}[j] \neq s[j]\}| \leq y,$$

where $Q^k = \{j \in Q : s_{i_1}[j] = s_k[j]\}$.

Proof. Dividing $d(s_k, s_{i_1}|Q)$ into two parts, we have

$$d(s_k, s_{i_1}|Q - Q^k) + d(s_k, s_{i_1}|Q^k) + d(s_k, s|P) = \frac{r}{r-1}d_b - y. \tag{7}$$

By Corollary 7 and with the further restriction that $s_l[j] = s[j]$, we have that, for any string $s_l \in \mathcal{B}$,

$$|\{j \in Q \mid s_{i_1}[j] \neq s_l[j] \text{ and } s_{i_1}[j] \neq s[j] \text{ and } s_l[j] = s[j]\}| \leq \frac{1}{r-1}\, d_b.$$

That is, for any string $s_l \in \mathcal{B}$,

$$|\{j \in (Q - Q^l) | s_l[j] = s[j]\}| \leq \frac{1}{r-1}\, d_b. \tag{8}$$

From (8), there exists β with $\beta \leq 1$ such that for any $s_l \in \mathcal{B}$, $|\{j \in (Q - Q^l) \mid s_l[j] = s[j]\}| \leq |\{j \in (Q - Q^k) \mid s_k[j] = s[j]\}| = \beta\frac{1}{r-1}\, d_b$. On the other hand, $|\{j \in (Q - Q^k) \mid s_k[j] \neq s_{i_1}[j]\}|$ is no more than $|\{j \in (Q - Q^k) \mid \}|$, which is the sum of $|\{j \in (Q - Q^k) \mid s_k[j] = s[j]\}|$ and $|\{j \in (Q - Q^k) \mid s_k[j] \neq s[j]\}|$. Combining those two formulae, we have

$$d(s_k, s|Q - Q^k) \geq d(s_k, s_{i_1}|Q - Q^k) - \beta\frac{1}{r-1}\, d_b. \tag{9}$$

Moreover, combining (7) and (9), we have

$$\begin{aligned} d(s_k, s|Q - Q^k) + \frac{\beta}{r-1}\, d_b + d(s_k, s_{i_1}|Q^k) + d(s_k, s|P) \\ \geq \frac{r}{r-1}d_b - y. \end{aligned} \tag{10}$$

Since

$$d(s_k, s|Q - Q^k) + d(s_k, s|Q^k) + d(s_k, s|P) \le d_b, \tag{11}$$

then (10) implies

$$\begin{aligned} d(s_k, s|Q^k) &\le d(s_k, s_{i_1}|Q^k) + \frac{\beta - 1}{r - 1} d_b + y \\ &= \frac{\beta - 1}{r - 1} d_b + y \le y. \end{aligned} \tag{12}$$

Here (12) is from the facts that $d(s_k, s_{i_1}|Q^k) = 0$ and $\beta \le 1$. Therefore,

$$|\{j \in Q^k | s_{i_1}[j] \ne s[j]\}| \le y. \tag{13}$$

This completes the proof. ∎

Here is our main lemma.

LEMMA 9. *For any constant r, there exist indices $1 \le i_1, i_2, \ldots, i_t \le n_1$, $r \le t \le 2r$, and a number $0 \le y \le d_b$ such that $|\{j \in Q_{i_1,i_2,\ldots,i_t} | s_{i_1}[j] \ne s[j]\}| \le y$. (Note that this implies $d(g, s_{i_1}|Q_{i_1,i_2,\ldots,i_t}) \ge d(g, s|Q_{i_1,i_2,\ldots,i_t}) - y$.) If we use s_{i_1} as the distinguishing string at positions in $Q_{i_1,i_2,\ldots,i_t}$, and s (the original distinguishing string) at the rest of the positions, then for any $s_l \in \mathcal{B}$, $d(s_l, s_{i_1}|Q_{i_1,i_2,\ldots,i_t}) + d(s_l, s|P_{i_1,i_2,\ldots,i_t}) \le \frac{r+1}{r-1} d_b - y$, where $P_{i_1,i_2,\ldots,i_t} = \{1, 2, \ldots, L\} - Q_{i_1,i_2,\ldots,i_t}$.*

Proof. We shall repeatedly apply Proposition 8 up to r times for the proof.

We start with $Q = Q_{i_1,i_2,\ldots,i_r}$. By Corollary 7, for any $s_l \in \mathcal{B}$, $d(s_l, s_{i_1}|Q) - |\{j \in Q \,|\, s_{i_1}[j] \ne s_l[j] \text{ and } s_{i_1}[j] = s_l[j]\}| \le \frac{1}{r-1} d_b$. Notice that $|\{j \in Q \,|\, s_{i_1}[j] \ne s_l[j] \text{ and } s_{i_1}[j] = s_l[j]\}| \le d(s_l, s \,|\, Q)$. We get

$$d(s_l, s_{i_1}|Q) + d(s_l, s|P)| \le \frac{1}{r - 1} d_b + d(s_l, s) \le \frac{r}{r - 1} d_b.$$

Therefore, there exist an index k (denoted by i_{r+1}) and number z^0 ($\frac{r}{r-1} d_b \ge z^0 \ge 0$) such that for any $s_l \in \mathcal{B}$,

$$d(s_l, s_{i_1}|Q) + d(s_l, s|P)| \le d(s_{i_{r+1}}, s_{i_1}|Q) + d(s_{i_{r+1}}, s|P)| = \frac{1}{r - 1} d_b - z^0.$$

By Proposition 8, $|\{j \in Q_{i_1,i_2,\ldots,i_r,i_{r+1}} | s_{i_1}[j] \ne s[j]\}| \le z^0$. Now apply Corollary 7 again to the index set $Q' = Q_{i_1,i_2,\ldots,i_{r+1}}$. Then, for any $s_l \in \mathcal{B}$, $d(s_l, s_{i_1}|Q_{i_1,i_2,\ldots,i_{r+1}}) + d(s_l, s|\{1, \ldots, L\} - Q_{i_1,i_2,\ldots,i_{r+1}}) \le \frac{r}{r-1} d_b$. Again, choose k' (and denote it by i_{r+2} later) and $y' \ge 0$ such that

$$d(s_l, s_{i_1}|Q_{i_1,i_2,\ldots,i_{r+1}}) + d(s_l, s|\{1, \ldots, L\} - Q_{i_1,i_2,\ldots,i_{r+1}}) \le \frac{r}{r - 1} d_b - y'$$

and such that equality holds for $l = k'$.

If $y' \ge z^0 - \frac{1}{r-1} d_b$, then our lemma follows with $t = r + 1$ and $y = z^0$. Therefore, we need to consider only the case $y' \le z^0 - \frac{1}{r-1} d_b$. Notice that if $z^0 < \frac{1}{r-1} d_b$, we will reach a contradiction here. We should also have $z^0 \ge \frac{1}{r-1} d_b$, or the lemma holds.

Define $z^1 = y'$. The conditions of Proposition 8 hold with $y = z^1$ and $Q = Q_{i_1,i_2,\ldots,i_{r+1}}$. Therefore, we are able to repeat the above process. Notice that $z^1 \le z^0 - \frac{1}{r-1} d_b$, and it is no smaller than $\frac{1}{r-1} d_b$. The same holds each time we repeat the

process. Therefore, the process can only be repeated up to r times before we obtain the result as claimed. □

In Lemma 9, we decompose the L positions into two parts, $Q_{i_1,i_2,\ldots,i_t}$ and $P_{i_1,i_2,\ldots,i_t}$. In $P_{i_1,i_2,\ldots,i_t}$, either the linear programming approach in section 2 or a brute-force enumeration method can go through since $|P_{i_1,i_2,\ldots,i_t}| \leq rd_b$. Details will be given in the next section. Thus, we can get a good approximation solution at positions in $P_{i_1,i_2,\ldots,i_t}$.

By Lemma 9, at positions in $Q_{i_1,i_2,\ldots,i_t}$ there are at most y positions where the letters for a feasible (optimal) solution are different from the letters in s_{i_1}. This implies that $d(g, s_{i_1}|Q_{i_1,i_2,\ldots,i_t}) \geq d(g, s|Q_{i_1,i_2,\ldots,i_t}) - y$. We may need to carefully choose y positions in $Q_{i_1,i_2,\ldots,i_t}$ in s_{i_1} to change to s'_{i_1} so that the condition $d(g, s'_{i_1}|Q_{i_1,i_2,\ldots,i_t}) \geq d(g, s|Q_{i_1,i_2,\ldots,i_t})$ is satisfied (with negligible error) for each $g \in \mathcal{G}$.

On the other hand, again by Lemma 9, if any y positions chosen in $Q_{i_1,i_2,\ldots,i_t}$ are changed to obtain a string s'_{i_1}, i.e, $d(s_{i_1}, s'_{i_1}) = y$, then $d(s_l, s|P_{i_1,i_2,\ldots,i_t}) + d(s_l, s'_{i_1}|Q_{i_1,i_2,\ldots,i_t}) \leq (1+\frac{1}{r-1})d_b$. That is, the total error produced for every string in $\mathcal{B}$ will be at most $\frac{1}{r-1}d_b$ no matter which y positions in $Q_{i_1,i_2,\ldots,i_t}$ are chosen. Therefore, to choose y positions in $Q_{i_1,i_2,\ldots,i_t}$, we can simply consider the good strings in $\mathcal{G}$ and ignore the strings in $\mathcal{B}$. This dramatically reduces the difficulty of the problem.

The main contribution of Lemma 9 is to transform the original DSP into the problem of how to choose y positions in $Q_{i_1,i_2,\ldots,i_t}$ to modify the string s_{i_1} into a string s' such that $d(g, s'_{i_1}|Q_{i_1,i_2,\ldots,i_t}) \geq d(g, s|Q_{i_1,i_2,\ldots,i_t})$. Though we do not know the value of y, we can enumerate all y: $y = 0, 1, \ldots, d_b$ in the algorithm. We will elaborate on this in the next section.

4. Choice of a distinguishing string. In general, we aim at constructing an approximate solution s'_{i_1} that differs from s_{i_1} at y positions (up to negligible error) in $Q_{i_1,i_2,\ldots,i_t}$. Though we do not know the value of y, we can enumerate all y: $y = 0, 1, \ldots, d_b$ in the algorithm. However, if $d_g \geq (r-1)y$, we can simply use s_{i_1} at all positions in $Q_{i_1,i_2,\ldots,i_t}$. For good strings $g \in \mathcal{G}$, the error at positions in $Q_{i_1,i_2,\ldots,i_t}$ will be $\frac{1}{r-1}d_g$, which will be good enough for a PTAS. For bad strings $s_l \in \mathcal{B}$, the error created by making s_{i_1} the approximate solution at positions in $Q_{i_1,i_2,\ldots,i_t}$ will be at most $\frac{2}{r-1}d_b$ (again, good enough for a PTAS) by Lemma 9. Therefore, we can assume that $d_g < (r-1)y$.

For any string $g_i \in \mathcal{G}$, if $d(g_i, s_{i_1}|Q_{i_1,i_2,\ldots,i_t}) \geq (r-1)y$, then the selected y positions in $Q_{i_1,i_2,\ldots,i_t}$ will cause error by at most $\frac{1}{r-1}d_g$. In fact, let s'_{i_1} be the string obtained from s_{i_1} by changing y positions in $Q_{i_1,i_2,\ldots,i_t}$. Then $d(g_i, s'_{i_1}|Q_{i_1,i_2,\ldots,i_t}) \geq (r-2)y$. Recalling the assumption that $d_g < (r-1)y$, we get for any $g_i \in \mathcal{G}$,

$$d(g_i, s'_{i_1}) \geq d(g_i, s'_{i_1}|Q_{i_1,i_2,\ldots,i_t}) \geq \frac{r-2}{r-1}d_g = \left(1 - \frac{1}{r-1}\right)d_g.$$

Therefore, we have to consider only the strings $g_i \in \mathcal{G}$ with the restriction that $d(g_i, s_{i_1}|Q_{i_1,i_2,\ldots,i_t}) < (r-1)y$. (This condition can be verified in polynomial time for each $g \in \mathcal{G}$.) In summary, without loss of generality, we have the following.

Assumption 1. For every string $g_i \in \mathcal{G}$, $d(g_i, s_{i_1}|Q_{i_1,i_2,\ldots,i_t}) < (r-1)y$ and $d_g \leq (r-1)y$.

In the remaining part of the section, several different methods will be used to carefully select y positions in $Q_{i_1,i_2,\ldots,i_t}$, each dealing with one of the three cases. Let $L' = |Q_{i_1,i_2,\ldots,i_t}|$. The complete algorithm is given in Figure 1.

In addition to the constant integer r, we need to introduce another positive number $\delta_0 > 0$. We should first choose r to be sufficiently large but remain a constant.

Algorithm gdistString

Input $\mathcal{B} = \{s_1, s_2, \ldots, s_{n_1}\}$ and $\mathcal{G} = \{g_1, g_2, \ldots, g_{n_2}\}$.
Output a center string s of length L.

1. **if** Case 1 holds, **then** use the method in section 2 to solve the problem.
2. **for** each $2r$-element subset $\{s_{i_1}, s_{i_2}, \ldots, s_{i_{2r}}\}$ of the n_1 input strings in $\mathcal{B}$ **do**
   (a) $Q = \{1 \le j \le L \mid s_{i_1}[j] = s_{i_2}[j] = \cdots = s_{i_t}[j]\}$, $P = \{1, 2, \ldots, L\} - Q$.
   (b) For the positions in $Q_{i_1,i_2,\ldots,i_t}$, use the methods described in Cases 2 and 3 to obtain a string of length $|Q_{i_1,i_2,\ldots,i_t}|$. Notice that the algorithm described in Case 1 produces directly a string s' of length L which is a PTAS.
   (c) Solve the optimization problem at the positions in $|P|$ using either an LP formulation (randomized rounding approach) (when $|P| \ge O(\log n)$) or an exhaustive search method (when $|P| \le O(\log n)$) to get an approximate solution x of length $|P|$.
3. Output the best solution obtained above.

FIG. 1. *Algorithm for DSP.*

Then we should choose δ_0 to be sufficiently small to achieve the required bound $(1+\epsilon)$ (and $(1-\epsilon)$) for PTAS. Notice that, since δ_0 is chosen after r is fixed, it may be a function of r. However, since r is a constant, we can make δ_0 sufficiently small (as a function of r) and at the same time δ_0 remains a constant. Therefore, in some of the asymptotic notation (Ω and big-O) used in this section, the constant may be a function of r. However, since we should set the value of r to a sufficiently large constant first, both r and δ_0 are constant when we later set the value of $\delta_0 > 0$ sufficiently small.

Case 1. $L' \le (r-1)^2 y$ and $y \ge (4\log(n_1+n_2))/{\delta_0}^2$. In this case, it follows that $d_b \ge y \ge (8\log(n_1+n_2))/{\delta_0}^2$. Moreover, we should prove that $d_b \ge c_b L$ and $d_g \ge c_g L$, where $c_b = c_g = \min\{\frac{1}{4r}, \frac{1}{2(r-1)^2}\}$.

Because $L' \ge y$ and $L' \le (r-1)^2 y$, it follows that $y = eL'$ for some $e : \frac{1}{(r-1)^2} \le e \le 1$. Since $d_b \ge y$ and $d_b \le d_g \le (r-1)y$ (from Assumption 1), we have $d_b = xy$ for $1 \le x \le r-1$. Thus, $d_b = xy = xeL'$ and $\frac{1}{(r-1)^2} \le xe \le r-1$. Note that $L = L' + |P|$ and $|P| \le td_b$ by Claim 4. We consider two subcases. (a) $|P| \le 0.5L$. Then $L' \ge 0.5L$. Thus $d_b = xeL' \ge 0.5xeL$. Since $d_b \le L$, we have $d_b = c'L$ for some $c' : \frac{1}{2(r-1)^2} \le c' \le 0.5(r-1)$. (b) $|P| > 0.5L$. Then by Claim 4, $d_b \ge |P|/t \ge 0.5L/t \ge L/4r$. Recall that we also assume that $d_g \ge d_b$ in the problem setting. It follows that $d_g \ge d_b > c_g L$.

Since $d_b \ge c_b L$ and $d_g \ge c_g L$, we can directly use the method described in section 2.

Case 2. $L' > (r-1)^2 y$ and $y \ge (4\log(n_1+n_2))/{\delta_0}^2$. By Assumption 1, $d(g_i, s_{i_1}|Q_{i_1,i_2,\ldots,i_t}) < (r-1)y$. Thus, for any $g_i \in \mathcal{G}$, s_{i_1} and g_i agree at most of the positions (at least $(r-1)(r-2)$ out of $(r-1)^2$ positions) in $Q_{i_1,i_2,\ldots,i_t}$. Intuitively, if we randomly select y positions in $Q_{i_1,i_2,\ldots,i_t}$ and change them into any different letters from s_{i_1} to get y_{i_1}, then y_{i_1} and g_i should differ at most of the y positions. Therefore, we can apply the following simple randomized algorithm:

1. Arbitrarily divide the L' positions into y sets of positions $Y_1, Y_2, \ldots, Y_y$ such that $|Y_1| = |Y_2| = \cdots = |Y_{y-1}| = \lfloor \frac{L'}{y} \rfloor$.

2. For each Y_i, independently randomly select a position in Y_i.
3. Let $Y = \{j_1, j_2, \ldots, j_y\}$ be the y selected positions in step 2, and y_{i_1} be a string of length $|Q_{i_1,i_2,\ldots,i_t}|$ obtained from s_{i_1} by changing the letters at these positions in Y (to any different letters) such that $d(y_{i_1}, s_{i_1}|Q_{i_1,i_2,\ldots,i_t}) = y$.

This algorithm gives us the needed solution for Case 2. We have the following proposition.

PROPOSITION 10. *The randomized algorithm for Case* 2 *produces a center string* y_{i_1} *of length* $|Q_{i_1,i_2,\ldots,i_t}|$ *with high probability such that*

(i) *for any* $s_i \in \mathcal{B}$, $d(y_{i_1}, s_i|Q_{i_1,i_2,\ldots,i_t}) + d(s, s_i|P_{i_1,i_2,\ldots,i_t}) \leq \frac{r+1}{r-1} d_b$, *and*

(ii) *for any* $g_j \in \mathcal{G}$, $d(y_{i_1}, g_j|Q_{i_1,i_2,\ldots,i_t}) + d(s, g_j|P_{i_1,i_2,\ldots,i_t}) \geq (1 - \frac{2}{r-1} - 2\delta_0) d_g$, *where* s *is a feasible (optimal) distinguishing string.*

Proof. (i) arises directly from Lemma 9.

Now, we show that (ii) is true for the case where $\lfloor \frac{L'}{y} \rfloor = \frac{L'}{y}$. (The general case is left to interested readers.) Consider a string $g \in \mathcal{G}$. Let Z_i be the set of positions j in Y_i with $s_{i_1}[j] = g[j]$. Let X_i be a 0/1 random variable. $X_i = 1$ indicates that j_i is in Z_i. The probability that $X_i = 1$ is $\frac{|Z_i|}{|Y_i|}$. Then we get

$$\mu = E\left[\sum_{i=1}^{y} X_i\right] = \sum_{i=1}^{y} \frac{|Z_i|}{|Y_i|} = \frac{y}{L'} \sum_{i=1}^{y} |Z_i| \geq \frac{y(L' - (r-1)y)}{L'}.$$

The last inequality is by the fact from Assumption 1 that $d(g, s_{i_1}|Q_{i_1,i_2,\ldots,i_t}) < (r-1)y$. Since $L' > (r-1)^2 y$, $\mu \geq \frac{y(r-2)}{r-1}$. By Lemma 1,

$$\mathbf{Pr}\left(\sum_{i=1}^{y} X_i < \mu - \delta_0 y\right) < \exp\left(-\frac{1}{2}\delta_0^2 y\right) \leq n_2^{-2}.$$

The last inequality is from the fact that $y \geq 4\log(n_1 + n_2)/\delta_0^2$.

Let y_{i_1} be the string obtained from s_{i_1} by changing the letters at y positions using the randomized algorithm. Then for the string $g \in \mathcal{G}$,

$$d(y_{i_1}, g|Q_{i_1,i_2,\ldots,i_t}) + \left(y - \sum_{i=1}^{y} X_i\right) - d(s_{i_1}, g|Q_{i_1,i_2,\ldots,i_t}) \geq \sum_{i=1}^{y} X_i.$$

The above inequality is due to the facts that (1) s_{i_1} and y_{i_1} differ at the y selected positions, (2) y_{i_1} and g differ at every position i with $X_i = 1$, and (3) y_{i_1} and g may agree at the rest of the $y - \sum_{i=1}^{y} X_i$ positions. Thus, we have

$$\frac{d(y_{i_1}, g|Q_{i_1,i_2,\ldots,i_t}) - d(s_{i_1}, g_l|Q_{i_1,i_2,\ldots,i_t}) + y}{2} \geq \sum_{i=1}^{y} X_i.$$

Therefore, we get

$$\begin{aligned} & \mathbf{Pr}\left(\frac{d(y_{i_1}, g|Q_{i_1,i_2,\ldots,i_t}) - d(s_{i_1}, g|Q_{i_1,i_2,\ldots,i_t}) + y}{2} < \mu - \delta_0 y\right) \\ \leq \quad & \mathbf{Pr}\left(\sum_{i=1}^{y} X_i < \mu - \delta_0 y\right) < \exp\left(-\frac{1}{2} y \delta_0^2\right) \leq n_2^{-2}. \end{aligned}$$

Substituting $\mu \geq \frac{y(r-2)}{r-1}$ into the above inequality, and after some calculation, we have

$$\mathbf{Pr}\left(d(y_{i_1}, g|Q_{i_1,i_2,\ldots,i_t}) - d(s_{i_1}, g|Q_{i_1,i_2,\ldots,i_t}) \leq \frac{y(r-3)}{r-1} - 2\delta_0 y\right) \leq n_2^{-2}.$$

From Lemma 9, $d(s_i, g|Q_{i_1,i_2,\ldots,i_t}) \geq d(s, g||Q_{i_1,i_2,\ldots,i_t}) - y$, so we get

$$\mathbf{Pr}\left(d(s, g|Q_{i_1,i_2,\ldots,i_t}) - d(y_{i_1}, g|Q_{i_1,i_2,\ldots,i_t}) > 2\delta_0 y + \frac{2y}{r-1}\right) \leq n_2^{-2}.$$

Considering all good strings, we get

$$\mathbf{Pr}\left(d(s, g|Q_{i_1,i_2,\ldots,i_t}) - d(y_{i_1}, g|Q_{i_1,i_2,\ldots,i_t}) > 2\delta_0 y + \frac{2y}{r-1} \text{ for any } g \in \mathcal{G}\right) \leq n_2^{-1}.$$

Therefore, with high probability we get that

$$d(y_{i_1}, g|Q_{i_1,i_2,\ldots,i_t}) \geq d(s, g|Q_{i_1,i_2,\ldots,i_t}) - \left(\frac{2}{r-1} + 2\delta_0\right) d_g,$$

that is,

$$\begin{aligned} & d(y_{i_1}, g|Q_{i_1,i_2,\ldots,i_t}) + d(s, g|P_{i_1,i_2,\ldots,i_t}) \\ \geq \quad & d(s, g_l) - \left(\tfrac{2}{r-1} + 2\delta_0\right) d_g \geq d_g - \left(\tfrac{2}{r-1} + 2\delta_0\right) d_g = \left(1 - \tfrac{2}{r-1} - 2\delta_0\right) d_g. \qquad \square \end{aligned}$$

Now, we give a derandomized algorithm for Case 2.

LEMMA 11. *There is a deterministic algorithm that runs in time equal to* $O(d_b^2(n_1+n_2)^{\log(2(r-1)^2e)8/\delta_0^2})$ *and finds a string* y_{i_1} *from* s_{i_1} *such that for any* $g \in \mathcal{G}$,

$$d(y_{i_1}, g|Q_{i_1,i_2,\ldots,i_{2r}}) - d(s_{i_1}, g|Q_{i_1,i_2,\ldots,i_{2r}}) \geq \frac{r-3}{r-1} y - 2\delta_0 y.$$

Proof. Proposition 10 holds when $L' = (r-1)^2 y$, since in that case $\mu = \frac{y(r-2)}{r-1}$. Therefore, if $L' > (r-1)^2 y$, we can arbitrarily select a subset $Q \subset Q_{i_1,i_2,\ldots,i_t}$ of size $(r-1)^2 y$ and try to select y positions in Q.

Next, if y is big enough, i.e., $y = (4\log(n_1+n_2))/{\delta_0}^2$, then Proposition 10 holds. That is, a good approximation solution, i.e., a set of y positions, exists in Q. We can try all possible ways to choose y positions in Q. The time required is $C^y_{(r-1)^2 y}$. Note that

$$C_m^y = \frac{m(m-1)\cdots(m-y+1)}{y!} \leq \frac{m^y}{y!} \leq \left(\frac{m}{y}\right)^y e^y.$$

We have

$$C^y_{(r-1)^2 y} \leq ((r-1)^2)^y e^y = (n_1+n_2)^{\log((r-1)^2 e)4/\delta_0^2}.$$

Thus, the time required is at most $O((n_1+n_2)^{\log((r-1)^2 e)4/\delta_0^2})$ if $y = (4\log(n_1 + n_2))/{\delta_0}^2$.

When $y > (4\log(n_1+n_2))/{\delta_0}^2$, we cannot directly choose y positions from Q since the time complexity is higher. Let $y = x \times (4\log(n_1+n_2))/{\delta_0}^2 + z$, where $x \geq 1$ is an integer and $z < (4\log(n_1+n_2))/{\delta_0}^2$. We divide Q into x disjoint subsets $Q_1, Q_2, \ldots, Q_x$ such that $|Q_i| = (r-1)^y/x$ for $i = 1, 2, \ldots, x-1$ and $Q_x = Q - Q_1 - Q_2 - \cdots - Q_{x-1}$. Proposition 10 ensures that each Q_i contains a good approximation solution, i.e., a set of $(4\log(n_1+n_2))/{\delta_0}^2$ positions, such that y_1 and g differ at most of the positions (at least $\frac{(r-2)}{r-1}\%$). Thus, we can try all possible ways to choose the right $(4\log(n_1+n_2))/{\delta_0}^2$ positions from each Q_i. (We have to choose

$(4\log(n_1+n_2))/{\delta_0}^2 + z$ positions from Q_x.) Therefore, the time required for each Q_i is $O((n_1+n_2)^{\log(2(r-1)^2e)8/\delta_0^2})$. Since there are at most y Q_i's and we have to try different values of y, the total time required is $O(d_b^2(n_1+n_2)^{\log(2(r-1)^2e)8/\delta_0^2})$. □

Case 3. $y < (4\log(n_1+n_2))/{\delta_0}^2$. For any q good strings $g_{j_1}, g_{j_2}, \ldots, g_{j_q}$, we define

$$R_{j_1,j_2,\ldots,j_q} = \{j \in Q_{i_1,i_2,\ldots,i_t} \mid s_{i_1}[j] \neq g_{j_l}[j] \text{ for at least one } l \in \{1,2,\ldots,q\}\}$$

and

$$\bar{R}_{j_1,j_2,\ldots,j_q} = Q_{i_1,i_2,\ldots,i_t} - R_{j_1,j_2,\ldots,j_q}.$$

From the definition of $R_{j_1,j_2,\ldots,j_q}$ and $\bar{R}_{j_1,j_2,\ldots,j_q}$, we have that for any integers p, q $(p<q)$,

$$R_{j_1,j_2,\ldots,j_p} \subseteq R_{j_1,j_2,\ldots,j_q}, \qquad \bar{R}_{j_1,j_2,\ldots,j_q} \subseteq \bar{R}_{j_1,j_2,\ldots,j_p}.$$

Let $y_1 = d(s_{i_1}, s|R_{j_1,j_2,\ldots,j_r})$ and $y_2 = d(s_{i_1}, s|\bar{R}_{j_1,j_2,\ldots,j_r})$. Then $y_1 + y_2 \leq y$.

The following lemma is important in dealing with Case 3.

LEMMA 12. *There exist* r *indices* $1 \leq j_1, j_2, \ldots, j_r \leq n_2$ *such that for any* $g \in \mathcal{G}$,

$$|\{j \in \bar{R}_{j_1,j_2,\ldots,j_r} \mid s_{i_1}[j] \neq g[j] \text{ and } s_{i_1}[j] \neq s[j]\}| \leq \frac{y}{r}.$$

Proof. Take $g_{j_1} \in \mathcal{G}$, and let

$$U_{j_1} = \{j \in Q_{i_1,i_2,\ldots,i_t} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g_{j_1}[j]\}$$

such that $|U_{j_1}|$ is as large as possible. Clearly, $U_{j_1} \subseteq R_{j_1}$.

If $|U_{j_1}| \leq \frac{y}{r}$, let $j_2, \ldots, j_r$ be any other indices and $g \in \mathcal{G}$ be any good string; it follows that

$$\begin{array}{ll} & |\{j \in \bar{R}_{j_1,j_2,\ldots,j_r} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g[j]\}| \\ \leq & |\{j \in Q_{j_1,j_2,\ldots,j_t} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g_{j_1}[j]\}| \\ = & |U_{j_1}| \leq \frac{y}{r}. \end{array}$$

Thus we may assume that $|U_{j_1}| > \frac{y}{r}$. Take $g_{j_2} \in \mathcal{G}$, and let

$$U_{j_1,j_2} = \{j \in Q_{i_1,i_2,\ldots,i_t} - U_{j_1} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g_{j_2}[j]\}$$

such that $|U_{j_1,j_2}|$ is as large as possible. Clearly, $U_{j_1,j_2} \subseteq R_{j_1,j_2}$.

If $|U_{j_1,j_2}| \leq \frac{y}{r}$, let $j_3, \ldots, j_r$ be any other indices and $g \in \mathcal{G}$ be any good string; it follows that

$$\begin{array}{ll} & |\{j \in \bar{R}_{j_1,j_2,\ldots,j_r} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g[j]\}| \\ = & |\{j \in Q_{1_1,i_2,\ldots,i_t} - R_{j_1,j_2,\ldots,j_r} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g[j]\}| \\ \leq & |\{j \in Q_{1_1,i_2,\ldots,i_t} - U_{j_1} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g_{j_2}[j]\}| \\ = & |U_{j_1,j_2}| \leq \frac{y}{r}. \end{array}$$

The inequality comes from the fact $U_{j_1} \subseteq R_{j_1} \subseteq R_{j_1,j_2,\ldots,j_r}$ and the choice of g_{j_2}. So we have to consider only $|U_{j_1,j_2}| > \frac{y}{r}$ since otherwise we are done.

In general, we can get a list of sets of positions

$$U_{j_1}, U_{j_1,j_2}, \ldots, U_{j_1,j_2,\ldots,j_q},$$

where $U_{j_1,j_2,\ldots,j_q} = \{j \in Q_{1_1,i_2,\ldots,i_t} - U_{j_1,j_2,\ldots,j_{q-1}} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g_{j_q}[j]\}$ and $|U_{j_1,j_2,\ldots,j_q}|$ is as large as possible such that

- $U_{j_1,j_2,\ldots,j_s} \cap U_{j_1,j_2,\ldots,j_t} = \emptyset$, $1 \leq s \neq t \leq q$,
- $|U_{j_1,j_2,\ldots,j_p}| > \frac{y}{r}$, $p = 1, \ldots, q$,
- q is as large as possible.

Note that $U_{j_1} \cup U_{j_1,j_2} \cup \cdots \cup U_{j_1,j_2,\ldots,j_q} \subseteq \{j \in Q_{i_1,i_2,\ldots,i_t} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g_{j_l}[j] \text{ for at least one } l \in \{1,2,\ldots,q\}\} \subseteq \{j \in Q_{i_1,i_2,\ldots,i_t} \mid s_{i_1}[j] \neq s[j]\}$. Therefore, we get

$$\frac{qy}{r} < |U_{j_1} \cup U_{j_1,j_2} \cup \cdots \cup U_{j_1,j_2,\ldots,j_q}| \leq |\{j \in Q_{i_1,i_2,\ldots,i_t} \mid s_{i_1}[j] \neq s[j]\}| \leq y,$$

and thus $q < r$.

Finally, take $g_{j_{q+1}} \in \mathcal{G}$, and let

$$U_{j_1,j_2,\ldots,j_{q+1}} = \{j \in Q_{i_1,i_2,\ldots,i_t} - U_{j_1,j_2,\ldots,j_q} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g_{j_{q+1}}[j]\}$$

such that $|U_{j_1,j_2,\ldots,j_{q+1}}|$ is as large as possible. Clearly, $U_{j_1,j_2,\ldots,j_{q+1}} \subseteq R_{j_1,j_2,\ldots,j_{q+1}}$.

By the maximality of q, $|U_{j_1,j_2,\ldots,j_{q+1}}| \leq \frac{y}{r}$, let $j_{q+2}, \ldots, j_r$ be any other indices and $g \in \mathcal{G}$ be any good string; it follows that

$$\begin{array}{rl} & |\{j \in \bar{R}_{j_1,j_2,\ldots,j_r} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g[j]\}| \\ = & |\{j \in Q_{i_1,i_2,\ldots,i_t} - R_{j_1,j_2,\ldots,j_r} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g[j]\}| \\ \leq & |\{j \in Q_{i_1,i_2,\ldots,i_t} - U_{j_1,j_2,\ldots,j_q} \mid s_{i_1}[j] \neq s[j] \text{ and } s_{i_1}[j] \neq g_{j_{q+1}}[j]\}| \\ = & |U_{j_1,j_2,\ldots,j_{q+1}}| \leq \frac{y}{r}. \end{array}$$

Lemma 12 is now proved. □

Now, we consider the two parts $R_{j_1,j_2,\ldots,j_r}$ and $\bar{R}_{j_1,j_2,\ldots,j_r}$, respectively.

Assumption 1 and the definition of $R_{j_1,j_2,\ldots,j_r}$ ensure that

$$\begin{array}{rcl} |R_{j_1,j_2,\ldots,j_r}| & = & |\{j \in Q_{1_1,i_2,\ldots,i_t} \mid s_{i_1}[j] \neq g_{j_l}[j] \text{ for at least one } l \in \{1,2,\ldots,r\}\}| \\ & \leq & \sum_{l=1}^{r} |\{j \in Q_{1_1,i_2,\ldots,i_t} \mid s_{i_1}[j] \neq g_{j_l}[j]\}| \\ & = & \sum_{l=1}^{r} d(s_{i_1}, g_{j_l}|Q_{1_1,i_2,\ldots,i_t}) \leq r(r-1)y < r^2 y. \end{array}$$

Keep in mind that we are dealing with the case $y \leq (4\log(n_1+n_2))/{\delta_0}^2$. Thus, we have $|R_{j_1,j_2,\ldots,j_r}| \leq (4\log(n_1+n_2))/{\delta_0}^2$. We can try all possible ways to achieve the optimal distinguishing string s at the positions in $R_{j_1,j_2,\ldots,j_r}$. The time required is

$$\sum_{y_1=0}^{y} C_{r^2 y}^{y_1} \leq 2^{r^2 y} \leq O((n_1+n_2)^{4r^2/\delta_0^2}).$$

Thus we can assume that we know the optimal distinguishing string s at the positions in $R_{j_1,j_2,\ldots,j_r}$.

Now, let us focus on the positions in $\bar{R}_{j_1,j_2,\ldots,j_r}$. For any $g \in \mathcal{G}$, define $Set(g, s_{i_1})$ be the set of positions in $\bar{R}_{j_1,j_2,\ldots,j_r}$, where g and s_{i_1} do not agree. From Lemma 12, we know that there exists a set of y_2 positions $K_{y_2} = \{k_j \in \bar{R}_{j_1,j_2,\ldots,j_r} \mid s_{i_1}[k_j] \neq s[k_j]\}$ such that for any $g \in \mathcal{G}$, $|K_{y_2} \cap Set(g, s_{i_1})| \leq \frac{y}{r}$. Though we do not know the exact value of y_2, again we can guess it in $O(y)$ time. Thus, we can assume that y_2 is known.

Let $s_{i_1}(K_{y_2})$ be the string obtained from s_{i_1} by changing the letters at the positions in K_{y_2} to any different letters. From Lemma 12, we can see that for any $g \in \mathcal{G}$,

$$d(g, s_{i_1}(K_{y_2})|\bar{R}_{j_1,j_2,\ldots,j_r}) \geq d(g, s|\bar{R}_{j_1,j_2,\ldots,j_r}) - \frac{1}{r}y. \tag{14}$$

Now, the only remaining task is to select y_2 positions in $\bar{R}_{j_1,j_2,\ldots,j_r}$ to approximate the set K_{y_2}. Recall that $|\{j \in \bar{R}_{j_1,j_2,\ldots,j_r} \mid s_{i_1}[j] \neq s[j]\}| = y_2$. From this we have that for every good string $g \in \mathcal{G}$,

$$d(g, s_{i_1}|\bar{R}_{j_1,j_2,\ldots,j_r}) - d(g, s|\bar{R}_{j_1,j_2,\ldots,j_r}) \geq -y_2. \tag{15}$$

Next we are going to show that we can find a set K'_{y_2} of y_2 positions in $\bar{R}_{j_1,j_2,\ldots,j_r}$ in polynomial time such that for any $g \in \mathcal{G}$, $|K'_{y_2} \cap Set(g, s_{i_1})| \leq \frac{y}{r}$. If this is true, by changing the letters of s_{i_1} on the positions in K'_{y_2}, the distance between $s_{i_1}(K'_{y_2})$ and g has to increase at least $y_2 - \frac{y}{r}$. To do this, let $m \leq |\bar{R}_{j_1,j_2,\ldots,j_r}|$ be an integer and $\bar{R}_{j_1,j_2,\ldots,j_r}(m) \subseteq \bar{R}_{j_1,j_2,\ldots,j_r}$ be any subset of $\bar{R}_{j_1,j_2,\ldots,j_r}$ with m elements.

The following two lemmas are the keys to solving the problem.

LEMMA 13. *If $m \geq 2^{r^2} \times y \times (n+1)^{\frac{r}{y}} + \frac{y}{r}$, then there exists a set $K'_{y_2} \subseteq \bar{R}_{j_1,j_2,\ldots,j_r}(m)$ of y_2 positions such that for any $g \in \mathcal{G}$,*

$$|K'_{y_2} \cap Set(g, s_{i_1})| \leq \frac{y}{r}.$$

Proof. Consider a string g in $\mathcal{G}$. Let $K''_{y_2} \subseteq \bar{R}_{j_1,j_2,\ldots,j_r}(m)$ denote a set of y_2 positions. By Assumption 1, that $d(s_{i_1}, g|Q_{i_1,i_2,\ldots,i_t}) < ry$, for any fixed $g \in \mathcal{G}$, the number of different K''_{y_2}'s such that

$$|K''_{y_2} \cap Set(g, s_{i_1})| \geq \frac{y}{r}$$

is at most

$$\begin{aligned} C_{m-ry}^{y_2-\frac{y}{r}} C_{ry}^{\frac{y}{r}} + C_{m-ry}^{y_2-\frac{y}{r}-1} C_{ry}^{\frac{y}{r}+1} + \cdots + 1 &\leq C_{m-ry}^{y_2-\frac{y}{r}} (C_{ry}^{\frac{y}{r}} + C_{ry}^{\frac{y}{r}+1} + \cdots + 1) \\ &\leq C_{m-ry}^{y_2-\frac{y}{r}} 2^{ry}. \end{aligned} \tag{16}$$

There are at most $n_2 \leq n_2 + n_1 = n$ strings in $\mathcal{G}$. Thus, if there is no $K''_{y_2} \subseteq \bar{R}_{j_1,j_2,\ldots,j_r}(m)$ such that

$$|K''_{y_2} \cap Set(g, s_{i_1})| \leq \frac{y}{r},$$

then

$$\begin{aligned} n_2 &\geq \frac{C_m^{y_2}}{2^{ry} C_{m-ry}^{y_2-\frac{y}{r}}} \\ &= \frac{m(m-1)\cdots(m-y_2+1)(y_2-\frac{y}{r})!}{y_2!(m-ry)(m-ry-1)\cdots(m-ry-y_2+\frac{y}{r}+1)2^{ry}} \\ &\geq \frac{m(m-1)\cdots(m-\frac{y}{r}+1)}{y_2(y_2-1)\cdots(y_2-\frac{y}{r}+1)2^{ry}} \\ &\geq \frac{(m-\frac{y}{r}+1)^{\frac{y}{r}}}{y^{\frac{y}{r}} \cdot 2^{ry}} > \frac{(m-\frac{y}{r})^{\frac{y}{r}}}{y^{\frac{y}{r}} 2^{ry}} = \frac{(m-\frac{y}{r})^{\frac{y}{r}}}{y^{\frac{y}{r}} (2^{r^2})^{\frac{y}{r}}} \\ &= \left(\frac{m-\frac{y}{r}}{2^{r^2} y}\right)^{\frac{y}{r}} \geq n+1. \end{aligned}$$

The last inequality is from the fact that $m \geq 2^{r^2} \times y \times (n+1)^{\frac{r}{y}} + \frac{y}{r}$. This is a contradiction. Thus, we can conclude that the lemma holds. ∎

Lemma 13 says that a good solution exists at the positions in $\bar{R}_{j_1,j_2,\ldots,j_r}(m)$ if $m = 2^{r^2} \times y \times (n+1)^{\frac{r}{y}} + \frac{y}{r}$. On the other hand, the following lemma shows that $m = 2^{r^2} \times y \times (n+1)^{\frac{r}{y}} + \frac{y}{r}$ is not too big so that we can find such a good solution in polynomial time by looking at all $C_m^{y_2}$ possible choices.

LEMMA 14. *If* $m = 2^{r^2} \times y \times (n+1)^{\frac{r}{y}} + \frac{y}{r}$ *and* $y \le (4\log(n_1+n_2))/{\delta_0}^2$, *then* C_m^y *is* $O((n_1+n_2)^{(4r^2+4+4\log e)/\delta_0^2+r}$.

Proof.

$$C_m^y = \frac{m(m-1)\cdots(m-y+1)}{y!} \le \frac{m^y}{y!}$$

$$\le \left(\frac{m}{y}\right)^y e^y \le \left(\frac{2(m-\frac{y}{r})}{y}\right)^y e^y \tag{17}$$

$$= \left(\frac{2(m-\frac{y}{r})}{2^{r^2}y}\right)^y (2^{r^2})^y e^y = 2^y (n+1)^r (2^{r^2})^y e^y. \tag{18}$$

Since $y \le (4\log(n_1+n_2))/{\delta_0}^2$, from (18), C_m^y is $O((n_1+n_2)^{(4r^2+4+4\log e)/\delta_0^2+r}$. ∎

LEMMA 15. *For Case* 3, *the time required to select* y *positions in* $Q_{i_1,i_2,\ldots,i+2r}$ *is* $O(rLd_b^2(n_1+n_2)^{(4r^2+4+4\log e)/\delta_0^2+2r})$.

Proof. The time to deal with $R_{j_1,j_2,\ldots,j_r}$ is smaller than that of $\bar{R}_{j_1,j_2,\ldots,j_r}$. It requires $O((n_1+n_2)^{(4r^2+4+4\log e)/\delta_0^2+r})$ time to deal with an $\bar{R}_{j_1,j_2,\ldots,j_r}$. Obtaining an $\bar{R}_{j_1,j_2,\ldots,j_r}$ needs $O(n_2^r Lr)$ time. We also have to guess y and y_2 (thus y_1) in d_b^2 time. Thus, the total time required to select y positions in $Q_{i_1,i_2,\ldots,i+2r}$ is $O(rLd_b^2(n_1+n_2)^{(4r^2+4+4\log e)/\delta_0^2+2r})$. ∎

THEOREM 16. *There is a PTAS for the DSP. The PTAS runs in* $O(L \times |\Sigma|^{O(\log(n_1+n_2)(r-1)^6)})$ *time and finds a distinguishing string* s *such that for every* $s_i \in \mathcal{B}$, $d(s_i,s) \le (1+\frac{1}{r-1})d_b$, *and for every* $g_i \in \mathcal{G}$, $d(g_i,s) \ge (1-\frac{1}{r-1})d_g$ *if a solution to the original pair* $(d_b \le d_g)$ *exists.*

Proof. According to Theorem 3, step 1 in Algorithm gdistString (Figure 1) requires $O(L \times |\Sigma|^{4\log(n_1+n_2)/\delta^2})$ time, where $\delta = \epsilon \min\{\frac{1}{4r}, \frac{1}{2(r-1)^2}\}$, and the performance ratios are $(1+\epsilon)$ and $(1-\epsilon)$ for bad strings and good strings, respectively.

Consider step 2 of Algorithm gdistString. We discuss Cases 2 and 3 separately.

Case 2. Lemma 11 shows that for a fixed $Q_{i_1,i_2,\ldots,i_{2r}}$ we can find a string y_{i_1} from s_{i_1} such that for each $g \in \mathcal{G}$ we have

$$d(y_{i_1}, g|Q_{i_1,i_2,\ldots,i_{2r}}) - d(s_{i_1}, g|Q_{i_1,i_2,\ldots,i_{2r}}) \ge \frac{r-3}{r-1}y - 2\delta_0 y$$

in $O(d_b^2(n_1+n_2)^{\log((r-1)^2e)8/\delta_0^2})$ time.

For the positions in $P_{i_1,i_2,\ldots,i_{2r}}$, we can use the LP approach to solve the following inequalities:

$$\begin{cases} \sum_{a\in\Sigma} x_{k,a} = 1, \quad k = 1,2,\ldots,|P_{i_1,i_2,\ldots,i_{2r}}|, \\ \sum_{k\in P_{i_1,i_2,\ldots,i_{2r}}} \sum_{a\in\Sigma} \chi(s,i,k,a)\, x_{k,a} \le \frac{r+1}{r-1} d_b - d(s_{i_1}, s_i|Q_{i_1,i_2,\ldots,i_{2r}}) - y, \\ \qquad i = 1,2,\ldots,n_1, \\ \sum_{k\in P_{i_1,i_2,\ldots,i_{2r}}} \sum_{a\in\Sigma} \chi(g,j,k,a)\, x_{k,a} \ge d_g - d(s_{i_1}, g_j|Q_{i_1,i_2,\ldots,i_{2r}}) - y, \\ \qquad j = 1,2,\ldots,n_2, \\ x_{k,a} \in \{0,1\}, \quad a \in \Sigma, \ k \in P_{i_1,i_2,\ldots,i_{2r}}. \end{cases} \tag{19}$$

Since $|P_{i_1,i_2,\ldots,i_{2r}}| \le 2rd_b$, from Theorem 3, the run time required is $O(2rd_b^2 \times |\Sigma|^{4\log(n_1+n_2)\times/\delta^2})$ for $\delta = \epsilon/(2r)$ since we have to guess the value of y. Since we

have to try all $Q_{i_1,i_2,\ldots,i_{2r}}$'s and the values of d_b and d_g, the total time required is $O(L^4 n_1^{2r}[(n_1+n_2)^{\log((r-1)^2 e)8/\delta_0^2} + r \times |\Sigma|^{4\log(n_1+n_2)\times/\delta^2}])$. The total errors are $(1+\epsilon+\frac{2}{r-1})d_b$ and $(1-\epsilon-\frac{2}{r}-2\delta_0)d_g$.

Case 3. Lemmas 13 and 15 show that we can find a string y_{i_1} in $O(rLy^2(n_1+n_2)^{(4r^2+4+4\log e)/\delta_0^2+2r})$ time such that for any $g \in \mathcal{G}$ we have $d(y_{i_1}, g|Q_{i_1,i_2,\ldots,i_{2r}}) \geq d(s, g|Q_{i_1,i_2,\ldots,i_{2r}}) - \frac{2}{r}$.

For positions in $P_{i_1,i_2,\ldots,i_{2r}}$, we do the same thing as in Case 2. The time required is $O(rd_b^2 \times |\Sigma|^{4\log(n_1+n_2)\times/\delta^2})$ for $\delta = \epsilon/(2r)$. Thus, the total time required is $O(rd^4 n_1^{2r}[L(n_1+n_2)^{(4r^2+4+4\log e)/\delta_0^2+2r} + |\Sigma|^{4\log(n_1+n_2)\times/\delta^2}])$. The total errors are $(1+\epsilon+\frac{2}{r-1})d_b$ and $(1-\epsilon-\frac{2}{r})d_g$ in this case.

Case 1 dominates the time complexity when the errors are small. For example, if $e = \frac{1}{r-1}$, then the algorithm finds a distinguishing string s such that for every $s_i \in \mathcal{B}$, $d(s_i, s) \leq (1+\frac{1}{r-1})d_b$; for every $g_i \in \mathcal{G}$, $d(g_i, s) \geq (1-\frac{1}{r-1})d_g$; and the running time is $O(L \times |\Sigma|^{O(\log(n_1+n_2)(r-1)^6)})$. □

5. The DSSP. In this section, we present the algorithm for the DSSP. The idea is to combine the sampling technique in [9] with the algorithm for the DSP. The difficulty here is that for each $s_i \in \mathcal{B}$, we do not know the substring t_i of s_i. The sampling approach in [9] is as follows: For any fixed $r > 0$, by trying all the choices of r substrings of length L from $\mathcal{B}$, we can assume that $t_{i_1}, t_{i_2}, \ldots, t_{i_r}$ are the r substrings of length L that satisfy Lemma 13 by replacing s_l with t_l and s_{i_j} with t_{i_j}. Let Q be the set of positions at which $t_{i_1}, t_{i_2}, \ldots, t_{i_r}$ agree and $P = \{1, 2, \ldots, L\} - Q$. By Lemma 13, $t_{i_1}|Q$ is a good approximation to $s|Q$. However, we do not know the letters at positions in P. Thus, we randomly pick $O(\log(mn))$ positions from P, where m is the length of bad strings. Suppose that the multiset of these random positions is R. By trying all length-$|R|$ strings, we can assume that we know $s|R$. Then for each $1 \leq i \leq n_1$ we find the substring t_i' from s_i such that $f(t_i') = d(s, t_i'|R) \times \frac{|P|}{|R|} + d(t_{i_1}, t_i'|Q)$ is minimized. Let s be the optimal distinguishing string. Then t_i denotes the substring of s_i that is closest to s. Let s^* be a string such that $s^*|P = s|P$ and $s^*|Q = t_{i_1}|Q$. Then [9] shows the following.

Fact 1. With probability $1 - ((nm)^{-2} + (nm)^{-\frac{4}{3}})$, $d(s^*, t_i') \leq d(s^*, t_i) + 2\epsilon|P|$ for all $1 \leq i \leq n$.

After obtaining a t_i' of s_i for every $s_i \in \mathcal{B}$, we have the DSP, which can be solved by using the algorithms developed in section 3.

THEOREM 17. *There is a polynomial time approximation scheme that takes $\epsilon > 0$ as part of the input and computes a center string s of length L such that for every $s_i \in \mathcal{B}$ there is a length-L substring t_i of s_i with $d(s_i, s) \leq (1+\epsilon)d_b$, and for every $g_i \in \mathcal{G}$, $d(g_i, s) \geq (1-\epsilon)d_g$ if a solution exists.*

Proof. Let s' be the solution obtained by our algorithm. Lemma 9 and Fact 1 ensure that for each $s_i \in \mathcal{B}$, $s(t_i', s') \leq \frac{r}{r-1}d_b + \epsilon'|P|$, where $\frac{1}{r-1}d_b$ is the error at the positions in Q and $\epsilon'|P|$ ($\epsilon' = 2\epsilon + \epsilon''$) is the total error at the positions in P produced by both the random sampling method and the algorithm for the DSP. Lemma 9, together with sections 3 and 4, ensures that for each $g \in \mathcal{G}$, $d(g, s') \geq (1-\epsilon)d_g$, where ϵ is the error rate introduced by the algorithm for the DSP.

The randomized algorithm can be derandomized by the standard methods; [7] gives a short and clear explanation. □

6. Discussion and remarks. Our work concludes the search for provably good algorithms for the DSSP by presenting a PTAS. Some techniques have been developed

here dealing with values of parameters smaller than $\log(n)$ but larger than $O(1)$. They may have other applications to similar problems.

REFERENCES

[1] A. Ben-Dor, G. Lancia, J. Perone, and R. Ravi, *Banishing bias from consensus sequences*, in Proceedings of the 8th Annual Combinatorial Pattern Matching Conference, Aarhus, Denmark, 1997, pp. 247–261.

[2] J. Dopazo, A. Rodríguez, J. C. Sáiz, and F. Sobrino, *Design of primers for PCR amplification of highly variable genomes*, Comput. Appl. Biosci., 9 (1993), pp. 123–125.

[3] L. Gąsieniec, J. Jansson, and A. Lingas, *Efficient approximation algorithms for the Hamming center problem*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99), Baltimore, MD, 1999, SIAM, Philadelphia, 1999, pp. S905–S906.

[4] M. Ito, K. Shimizu, M. Nakanishi, and A. Hashimoto, *Polynomial-time algorithms for computing characteristic strings*, in Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, Asilomar, CA, 1994, pp. 274–288.

[5] J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang, *Distinguishing string selection problems*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99), Baltimore, MD, 1999, SIAM, Philadelphia, 1999, pp. 633–642.

[6] M. Li, B. Ma, and L. Wang, *Finding similar regions in many strings*, in Proceedings of the 31st ACM Symposium on Theory of Computing, Atlanta, GA, 1999, pp. 473–482.

[7] M. Li, B. Ma, and L. Wang, *On the closest string and substring problems*, J. Assoc. Comput. Mach., 49 (2002), pp. 157–171.

[8] K. Lucas, M. Busch, S. Mössinger, and J.A. Thompson, *An improved microcomputer program for finding gene- or gene family-specific oligonucleotides suitable as primers for polymerase chain reactions or as probes*, Comput. Appl. Biosci., 7 (1991), pp. 525–529.

[9] B. Ma, *A polynomial time approximation scheme for the closest substring problem*, in Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching, Montreal, Canada, 2000, pp. 99–107.

[10] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[11] C.H. Papadimitriou and M. Yannakakis, *On the approximability of trade-offs and optimal access of web sources*, in Proceedings of the 41st Annual Symposium on Foundations of Computer Science, Redondo Beach, CA, 2000, IEEE, pp. 86–92.

[12] V. Proutski and E. C. Holme, *Primer master: A new program for the design and analysis of PCR primers*, Comput. Appl. Biosci., 12 (1996), pp. 253–255.

SIAM J. COMPUT.
Vol. 32, No. 4, pp. 1091–1113

WINDOWS SCHEDULING PROBLEMS FOR BROADCAST SYSTEMS*

AMOTZ BAR-NOY† AND RICHARD E. LADNER‡

Abstract. The *windows scheduling problem* is defined by the positive integers n, h, and $w_1, \ldots, w_n$. There are n pages where the *window* w_i is associated with *page* i, and h is the number of *slotted channels* available for broadcasting the pages. A schedule that solves the problem assigns pages to slots such that the gap between any two consecutive appearances of page i is at most w_i slots. We investigate two optimization problems. (i) The *optimal windows scheduling problem*: given $w_1, \ldots, w_n$ find a schedule in which h is minimized. (ii) The *optimal harmonic windows scheduling problem*: given h find a schedule for the windows $w_i = i$ in which n is maximized. The former is a formulation of the problem of minimizing the bandwidth in push systems that support guaranteed delay, and the latter is a formulation of the problem of minimizing the startup delay in media-on-demand systems. For the optimal windows scheduling problem we present an algorithm that constructs asymptotically close to optimal schedules, and for the optimal harmonic windows scheduling problem we show how to achieve the largest known n's for all values of h.

Key words. scheduling algorithms, windows scheduling, broadcast systems, media-on-demand systems, push systems

AMS subject classification. 68W05

DOI. 10.1137/S009753970240447X

1. Introduction.

The *windows scheduling problem* is defined as follows, using terminology borrowed from a broadcasting system environment:

> Given are h slotted channels and n pages, $1, \ldots, n$, each associated with a window $w_i \geq 1$ that is an integral number. Is it possible to schedule the n pages on the h channels, one page per one channel at each time slot, such that the gap between two consecutive appearances of page i is no more than w_i?

If for a given h and $W = \langle w_1, \ldots, w_n \rangle$ the answer to the windows scheduling problem is positive, the schedule that solves the problem is called an $\langle h, W \rangle$-schedule. The natural optimization problem is to find for a given W the minimum h, denoted by $H(W)$, such that there exists an $\langle h, W \rangle$-schedule. We call this problem the *optimal windows scheduling problem.* Since a restricted version of the optimal windows scheduling problem is NP-hard even for one channel [3], we will be looking for approximation solutions.

As an example of a windows scheduling problem consider the vector $W = \langle 2, 4, 5 \rangle$, which can be scheduled on one channel as follows:

$$(1) \qquad \left[\begin{array}{cccccccccc} 2 & 4 & 2 & 5 & 2 & 4 & 2 & 5 & \cdots \end{array} \right].$$

*Received by the editors March 22, 2002; accepted for publication (in revised form) February 20, 2003; published electronically July 11, 2003. A preliminary version of this paper appeared in *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'02, SIAM, Philadelphia, pp. 433–442.

http://www.siam.org/journals/sicomp/32-4/40447.html

†Computer and Information Science Department, Brooklyn College, 2900 Bedford Avenue, Brooklyn, NY 11210 (amotz@sci.brooklyn.cuny.edu). This work was done in part while this author was a member of AT&T Labs-Research, Shannon Lab, Florham Park, NJ.

‡Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195 (ladner@cs.washington.edu). This work was done in part while this author visited AT&T Labs-Research, Shannon Lab, Florham Park, NJ, and was partially supported by NSF grants CCR-9732828 and CCR-0098012.

Note that 2 is scheduled every 2 slots and 4 is scheduled every 4 slots, but 5 is scheduled every 4 slots. Clearly, this problem cannot be solved in any fewer number of channels so that $H(W) = 1$.

The *harmonic windows scheduling problem* is a special case of the windows scheduling problem in which $w_i = i$ for $1 \leq i \leq n$. If for given h and n the answer to the harmonic windows scheduling problem is positive, then the schedule that solves the problem is called an $\langle h, n\rangle$-schedule. Here, for a given n the minimum h such that there exists an $\langle h, n\rangle$-schedule is denoted by $H(n)$. However, an optimization objective that is more natural to the harmonic windows scheduling problem is to find for a given h the maximum n, denoted by $N(h)$, such that there exists an $\langle h, n\rangle$-schedule. We call this problem the *optimal harmonic windows scheduling problem*.

It is not difficult to see that $N(1) = 1$ and $N(2) = 3$ using the $\langle 2, 3\rangle$-schedule

$$\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots \\ 2 & 3 & 2 & 3 & \cdots \end{bmatrix}. \tag{2}$$

A more interesting example is the $\langle 3, 9\rangle$-schedule demonstrating that $N(3) \geq 9$:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdots \\ 2 & 4 & 2 & 5 & 2 & 4 & 2 & 5 & 2 & 4 & 2 & 5 & \cdots \\ 3 & 6 & 7 & 3 & 8 & 9 & 3 & 6 & 7 & 3 & 8 & 9 & \cdots \end{bmatrix}. \tag{3}$$

This $\langle 3, 9\rangle$-schedule is known from the literature [15], but it is unknown whether the schedule is optimal. There may be a $\langle 3, 10\rangle$-schedule, but there is no $\langle 3, 11\rangle$-schedule (as implied by Theorem 3). Note also that the windows scheduling problem $W = \langle 2, 4, 5\rangle$ has a solution as the second channel of the $\langle 3, 9\rangle$-schedule above. Thus, the windows scheduling problem is not only a generalization of the harmonic windows scheduling problem, but solutions to windows scheduling problems can be merged together to solve a harmonic windows scheduling problem.

We note that the performance criterion for these two optimization problems is based on the *max* metric and not the *average* metric. That is, the next appearance of a page depends only on its previous appearance. In the traditional scheduling problems the focus was on the average metric, where in a way the next appearance of a page depends on all of its previous appearances.

The optimal windows scheduling problem is a formulation of the problem of minimizing the bandwidth of push systems that support guaranteed delay for the broadcasted pages. The optimal harmonic windows scheduling problem is an alternative formulation of the problem of minimizing the guaranteed startup delay in recently proposed media-on-demand systems.

1.1. The push systems application. In *push systems*, servers broadcast pages over broadcast channels to clients. Clients who wish to access one of the broadcasted pages listen to the channels until they receive this page. In traditional push systems (e.g., the Teletext problem [2] and the broadcast disks problem [1]), the server broadcasts more popular pages more frequently to minimize the average access time for clients. In some situations, the server does not have this freedom since it broadcasts information of some providers who "paid" for a certain quality of service.

These situations were considered by [7, 4]. In their push systems model they assume that the broadcasting environment is composed of three groups of "players": (i) Clients who wish to access information pages from broadcast channels. (ii) Servers who broadcast the information pages on channels. (iii) Providers who supply the information pages. Servers "sell" quality of service to providers who "pay" proportionally

to the inverse of the maximum gap (access time) between any two appearances of their page on any of the channels.

After the selling process, the server is left with the optimal windows scheduling problem. That is, the server tries to minimize the number of channels that are needed to guarantee the quality of service "bought" by the providers.

1.2. The media-on-demand application. Media-on-demand is the demand by clients to play back, view, listen, or read various types of media such as video, audio, or large files with startup delays as small as possible and with no interruptions. The solution of dedicating a private channel to each client for the required media is implausible even with the ever-growing availability of network bandwidth. Thus, multicasting popular media to groups of clients seems to be the ultimate solution to the ever-growing demand for media. The first, and most natural, idea to exploit the advantage of multicasting is to batch clients together. This implies a tradeoff between the overall server bandwidth and the guaranteed startup delay. The main advantage of the batching solutions lies in their simplicity. The main disadvantage is that the guaranteed startup delay may still be too large.

The pyramid broadcasting paradigm, pioneered by Viswanathan and Imielinski [17], was the first solution that reduced dramatically the bandwidth requirements for servers by using a larger receiving bandwidth for clients and by adding buffers to clients. The novelty of the pyramid broadcasting paradigm is that clients with more complicated equipment are able to receive more bandwidth than they need for playback. This way they can buffer later portions of the required transmission to play them back on time. Many papers followed this line of research with various models. All of them demonstrated that with added complexity to the system, a huge improvement can be achieved over the traditional batching solutions.

To demonstrate the huge gain, suppose first that a server dedicates two channels to a two-hour movie. In traditional systems, by staggering the transmissions over the two channels, the server can guarantee a one-hour startup delay. In systems with the new paradigm, the server can guarantee a startup delay of 40 minutes. Now suppose that a server dedicates 12 channels to a two-hour movie. In this case, the improvement will be from ten minutes to less than two seconds with a simple solution using the new paradigm and to less than one-tenth of a second with a more complicated solution found in this paper.

In this paper, we adopt the model of [9, 11] that focuses on the tradeoff between the server bandwidth and the guaranteed startup delays and ignores the other two parameters, the receiving bandwidth of clients and their buffer size. In this model, the transmission is divided into n equal-size segments and the time is slotted with slots of size $1/n$ of the transmission length. Servers broadcast the n segments on h channels according to some transmission schedule. Clients who wish to receive the transmission wait for a beginning of a slot, and thereafter they buffer the first appearance of segment i for $1 \leq i \leq n$. If for all $1 \leq i \leq n$ segment i appears in one of the ith slots after the client arrival time, then the client is guaranteed an uninterrupted playback. This is because when the client needs to play back the ith segment, it is either in its buffer or transmitted on one of the channels. It is assumed that clients have large enough buffers that can store a full transmission and that they can receive data from all the channels concurrently.

We demonstrate this model with the simple case of a server with two channels (see schedule (1) in the introduction). The transmission is divided into three segments. The first segment is broadcasted repeatedly on the first channel. The second and

third segments are broadcasted alternately on the second channel. A client has two possible starting times for receiving the transmission. The first is before a slot in which the second segment is broadcasted on the second channel and the second is before a slot in which the third segment is broadcasted on the second channel. (i) In the first time slot, in both cases, the client plays back the first segment from the first channel. At the same time, in the first case the client buffers segment two, and in the second case the client buffers segment three. (ii) In the second time slot, in the first case the client plays back segment two from its buffer while buffering segment three. In the second case, the client plays back segment two from the second channel. (iii) In the third time slot, in both cases, the client plays back segment three from its buffer. In this example, the guaranteed startup delay is one-third the transmission length.

A necessary and sufficient condition for clients to view the transmission with no interruptions is that segment i is broadcasted at least once in any consecutive i slots. In particular, a channel must be dedicated to the first segment, and the second segment must appear in at least every other slot on one of the other channels. Thus, if an $\langle h, n\rangle$-schedule exists, then clients can view the movie with no interruption with a guaranteed startup delay of $1/n$ of the movie length. To see this, assume first that clients arrive only at the beginning of slots; in this case a client may view the movie immediately with no interruptions. This is because the ith segment will be either on one of the channel's i time slots after the arrival time of the client or in the client's buffer. As a result, clients who arrive in arbitrary times need to wait for a beginning of a slot in order to view the movie with no interruptions. The maximum startup delay is therefore $1/n$ of the movie length.

Remark. A "fractional" version of the problem allows mixing segments together in each channel. With this capability, the papers [10, 13, 14] present solutions that almost match the lower bound of [6]. However, this fractional model does not apply to the general windows scheduling problem and to the push systems application. Furthermore, even for the media-on-demand application, in many systems the "integral" version of the problem is more realistic.

1.3. Previous results and our contributions. To the best of our knowledge there are no published solutions for the optimal windows scheduling problem as formulated in this paper. Papers [8, 11] present a simple $\langle h, 2^h - 1\rangle$-schedule for the harmonic windows scheduling problem. The best known results for the harmonic windows scheduling problem appear in [15, 12]. The pagoda scheme [15] starts with a $\langle 3, 9\rangle$-schedule and then generalizes this schedule for $h > 3$ as follows. For an even $h > 3$, the Pagoda scheme is an $\langle h, (4/5)\sqrt{5}^h - 1\rangle$-schedule, and for an odd $h > 3$, it is an $\langle h, (2/\sqrt{5})\sqrt{5}^h - 1\rangle$-schedule. Asymptotically, the pagoda scheme implies an $N(h) = O(\sqrt{5}^h) \approx O(2.236^h)$ solution. The new pagoda scheme [12] deals with small values of h. Their lower bounds on $N(h)$ for $h = 3, 4, 5, 6, 7$ are $9, 26, 66, 172, 442$, respectively (see Table 1).

We first present an asymptotic result for the optimal windows scheduling problem. Define $h(W) = \sum_{i=1}^{n} 1/w_i$. A simple argument shows that $\lceil h(W)\rceil$ is a lower bound for $H(W)$ since page i requires at least a $1/w_i$ portion of a channel. We show that a schedule exists that uses at most $h(W) + e\ln(h(W)) + 7.3595$ channels.[1] In other words, this schedule is within a factor of $1 + O\left(\ln(h(W))/h(W)\right)$ of optimal. This translates into a solution with $\left(e - O\left(\ln(h)/h\right)\right)^h$ pages for the harmonic windows

[1] The symbol e is the base of the natural logarithm ln. All decimal constants are at the precision provided by the software package *Mathematica* for a predetermined number of decimal points [18].

scheduling problem where h is the number of given channels and $0.561463e^h$ is an upper bound for $N(h)$.

The second part of our contribution focuses on "practical" approximate solutions to the optimal harmonic windows scheduling problem. Recall that in this case we fix the number of channels h and seek to find an $\langle h, n\rangle$-schedule for the largest possible n. An optimal solution would be an $\langle h, N(h)\rangle$-schedule. The asymptotic result implied by the solution to the optimal windows scheduling problem does not generate schedules "good" enough for small h. Most of the known constructions are based on more restrictive schedules. For example, the $\langle h, 2^h - 1\rangle$-schedule (see section 2) schedules each page at fixed intervals, schedules a page only on one channel, and on each channel schedules only consecutive pages. In this paper, all of the constructions schedule each page on only one channel. Most of them will schedule a page at fixed intervals. We call such schedules *perfect schedules*.

We first describe a greedy schedule that approaches the upper bound for $N(h)$ for small values of h. This greedy scheme is based on a tree representation of perfect schedules introduced in [5]. The greedy strategy can be applied to the general windows scheduling problem as well to obtain good bounds for $H(W)$ for cases in which $h(W)$ is small. Next we develop a technique that combines an $\langle h_1, n_1\rangle$-schedule with an $\langle h_2, n_2\rangle$-schedule to get an $\langle h_1 + h_2, ((n_1 + 1)(n_2 + 1))/2 - 1\rangle$-schedule. With this technique, we get the best results for the number of channels for which we cannot run the greedy algorithm. Finally, we present our nontrivial nonperfect schedules for $h = 4, 5, 6$ channels. These schedules were constructed using the tree representation technique with some modifications to accommodate nonperfect schedules.

TABLE 1
Results for 1 *to* 12 *channels.*

# of channels	1	2	3	4	5	6	7	8	9	10	11	12
Upper bound	1	3	10	30	82	226	615	1673	4549	12366	33616	91379
Best bound	1	3	9	28	77	211	570	1573	4325	11759	31677	86428
Greedy	1	3	9	25	73	201	565	1522	4284	11637	31677	86428
Pagoda	1	3	9	26	66	172	442	499	1249	2499	6249	12499

We summarize the bounds for 1 to 12 channels in Table 1. For a given h, we compare four values for $N(h)$: the upper bound, our best lower bound, the bound of the basic greedy algorithm described in section 4, and the best lower bound of the pagoda scheme [15] and the new pagoda scheme [12]. Our best bounds for small h $(4, 5, 6)$ were achieved by hand tuning the results of the greedy algorithm and for larger h by modifications to the greedy algorithm. To interpret Table 1 imagine that we can devote 8 channels to a two-hour movie. Without any of these schemes a new movie can start every $120/8 = 15$ minutes. With the pagoda scheme a movie can start every $7200/499 \approx 14.5$ seconds. With our best bound a movie can start every $7200/1573 \approx 4.6$ seconds. No such scheme can guarantee starting any faster than every $7200/1673 \approx 4.3$ seconds.

1.4. Organization. Section 2 presents some definitions and notation used later in the paper. Section 3 presents the upper bound and the asymptotic lower bounds for the optimal windows scheduling problem and the optimal harmonic windows scheduling problem. Section 4 describes the greedy algorithm which is an approximation algorithm for the optimal harmonic windows scheduling problem. Section 5 presents the combination technique for the optimal harmonic windows scheduling problem. Section 6 illustrates our best solutions for small numbers for the harmonic windows

scheduling problem. Finally, section 7 discusses some work in progress and open problems.

2. Preliminaries. In subsection 2.1, we present some of our definitions in a more formal way. In subsection 2.2, we describe a tree representation for perfect schedules that does not cover all the possible perfect schedules but is enough to serve as a base for our greedy algorithm and to represent most of our solutions for small numbers. The idea is borrowed from [5]. We describe what is needed to understand this paper.

2.1. Definitions and notation. A more formal definition of the windows scheduling problem and its special case, the harmonic windows scheduling problem, is as follows. We are asked to create an $h \times \infty$ matrix C in which the entry $C(j,t)$ is the page number that is scheduled at time slot t on the jth channel for $1 \le j \le h$ and $1 \le t \le \infty$. The requirement from the matrix is that for any $1 \le i \le n$ and $1 \le t \le \infty$, i is an element in the submatrix $C[1..h, t..t + w_i - 1]$. That is, i appears in one of the entries in the following submatrix:

$$\begin{bmatrix} C(1,t) & \dots & C(1,t+w_i-1) \\ \vdots & & \vdots \\ C(h,t) & \dots & C(h,t+w_i-1) \end{bmatrix}.$$

For the harmonic windows scheduling problem the same definition holds where $w_i = i$. Although a schedule is formally defined by an infinite matrix, any reasonable solution will be defined as the infinite concatenation of a finite schedule C which is a finite $h \times \ell$ matrix for some ℓ. Although it is not required, in all of our solutions each page is scheduled on a single channel. Such a solution has the property that for each page i, there is a channel h_i such that $i \notin \{C(h',t) : h' \ne h_i \text{ and } 1 \le t \le \ell\}$. Because of this we can focus on the schedules for individual channels.

Let $W = \langle w_1, \dots, w_n \rangle$ be a window vector. For a sequence $S = \langle s_0, \dots, s_{\ell-1} \rangle$ with $s_i \in \{0, 1, \dots, n\}$, define $M_S \subseteq \{1, \dots, n\}$ as the set of nonzero numbers in the sequence S. The sequence S is a potential row of a finite schedule for W. If $s_i = 0$, then no page is assigned to position i in the schedule. We define $S = \langle s_0, \dots, s_{\ell-1} \rangle$ to be a *channel schedule* for W if for each $i \in M_S$, if $i(1) < i(2) < \cdots < i(n_i)$ are all the indices such that $s_{i(j)} = i$, then $i(j+1) - i(j) \le w_i$ for $1 \le j < n_i$ and $\ell - i(n_i) + i(1) \le w_i$. This is equivalent to saying that when we repeat the schedule infinitely, for each i, i appears in the infinite schedule in every consecutive w_i slots. A channel schedule S is *perfect* if there exists a $w'_i \le w_i$ such that $i(j+1) - i(j) = w'_i$ for $1 \le j < n_i$ and $\ell - i(n_i) + i(1) = w'_i$. We call w'_i the window size of i in the perfect channel schedule S. If we repeat a perfect schedule infinitely, then in the infinite schedule, for each i, i appears periodically every w'_i slots. A channel schedule S is *fully utilized* if $s_i \ne 0$ for all i, that is, every position in the schedule has some page assigned to it. The following proposition should be clear.

PROPOSITION 1. *If S is a (perfect) channel schedule, then so is the result of a finite concatenation of S with itself, a cyclic shift of S, and the reversal of S.*

A finite schedule for W can be formed from h channel schedules $S_1, \dots, S_h$ of lengths $\ell_1, \dots, \ell_h$, respectively, where $\cup_{i=1}^{h} M_{S_i} = \{1, 2, \dots, n\}$. Let ℓ be the least common multiple of $\{\ell_1, \dots, \ell_h\}$. By a finite concatenation of each channel schedule with itself, S_i can be transformed into a channel schedule of length ℓ. These channel schedules form the rows of a finite schedule matrix. A *perfect schedule* for W is one

formed from perfect channel schedules. We define a perfect $\langle h, n\rangle$-schedule to be a perfect schedule for the window vector $\langle 1, 2, \ldots, n\rangle$ using h channels.

2.2. A tree representation for perfect channel schedules. Let T be an ordered tree with n labeled leaves. Assume first that all the labels (pages) are distinct. We describe a procedure that generates a perfect channel schedule on the n pages from T, computes the length of this channel schedule, and computes the window size of each page.

PROCEDURE TREE-TO-SCHEDULE.

Input: A tree T with n leaves that are labeled with the labels $p_1, \ldots, p_n$ that represent the pages. Assume T has $d \geq 0$ subtrees.

Output: A perfect channel schedule S for these pages in which each page p_i has a fixed window size w'_i.

A leaf tree: If $n = 1$, then the schedule is $S = \langle p_1\rangle$.

The recursive step: If $n > 1$ and therefore $d > 0$, do the following:

- Recursively construct the channel schedules $S_1, \ldots, S_d$ of all the d subtrees of T. Assume their respective lengths are $\ell_1, \ldots, \ell_d$.
- Replicate each channel schedule and make all of them have the same length $\ell' = LCM\{\ell_1, \ldots, \ell_d\}$. Let the new schedules be $S'_1, \ldots, S'_d$. By Proposition 1, a replication of a perfect channel schedule results in a perfect channel schedule.
- The final channel schedule S of length $\ell = d\ell'$ is constructed by alternately picking ℓ' times the next page from the d channel schedules $S'_1, \ldots, S'_d$.

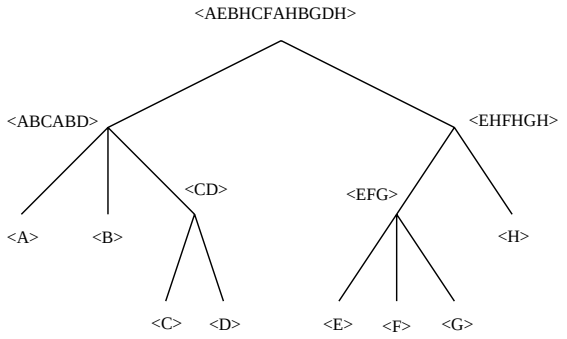


FIG. 1. *An example of a tree for pages* A, B, C, D, E, F, G, H.

A simple example is the round-robin tree. This is a tree whose root has d children, all of them leaves. If they are labeled by the numbers $p_1, p_2, \ldots, p_d$, then the channel schedule represented by this tree is the round-robin channel schedule $S = \langle p_1, p_2, \ldots, p_d\rangle$. A more complicated example is the tree in Figure 1. The labels are $A, B, \ldots, H$. Applying Procedure Tree-to-Schedule on the left and right subtrees yields the channel schedules $\langle ABCABD\rangle$ and $\langle EHFHGH\rangle$, respectively. Hence, the schedule representation of the tree is $\langle AEBHCFAHBGDH\rangle$.

We now give the condition for when Procedure Tree-to-Schedule generates a perfect channel schedule. Let $W = \langle w_1, \ldots, w_n\rangle$ be a window vector and let T be a tree with leaf labels $L \subseteq \{1, \ldots, n\}$. Let $W' = \cup_{i\in L}\{w_i\}$ be the window vector W restricted to the labels L. For $i \in L$, define $w'_i = \prod_{j=1}^{k} d_j$, where $d_1, \ldots, d_k$ are the

degrees of all the ancestors of the leaf labeled i in T from the root to the parent of the leaf labeled i. That is, d_1 is the degree of the root and d_k is the degree of the parent of the leaf labeled i. Let $\ell = LCM\{w'_1, \ldots, w'_{|L|}\}$.

THEOREM 2. *Procedure Tree-to-Schedule always produces a perfect channel schedule. This schedule satisfies W' if $w'_i \le w_i$ for all $i \in L$. In this case, w'_i is the window size of i in a channel schedule of length ℓ.*

Proof. We prove by induction on $|L|$ that the sequence $S = \langle s_0, \ldots, s_{\ell-1} \rangle$ produced by Procedure Tree-to-Schedule has the following *perfect* property:

> for each $i \in L$, if $i(1) < i(2) < \cdots < i(n_i)$ are all the indices such that $s_{i(j)} = i$, then $i(j+1) - i(j) = w'_i$ for $1 \le j < n_i$ and $\ell - i(n_i) + i(1) = w'_i$.

The theorem follows immediately from this.

For trees with one leaf this is clearly true. If the root of T has d_1 children, then by the induction hypothesis the procedure run on each of the d_1 subtrees produces the sequences $S_1, \ldots, S_{d_1}$, each with the perfect property. It follows that $S'_1, \ldots, S'_{d_1}$ defined by the procedure also have the perfect property because each is just a replication of a sequence with the perfect property (Proposition 1). In the final schedule S produced by the procedure, the label i occurs once every $w'_i = d_1 w''_i$ slots, where $w''_i = \prod_{j=2}^{k} d_j$ and $d_2, \ldots, d_k$ are the degrees of ancestors of the leaf labeled i in the subtree of T that contains i. Hence, the sequence S satisfies the perfect property. Finally, it is not hard to verify from the definition of the procedure that the length of S is $\ell = LCM\{w'_1, \ldots, w'_{|L|}\}$. □

Remark. So far we assumed that all the labels are distinct. This is not imperative. However, if two labels are the same, then the window size of the corresponding page may get more than one value. We use nonperfect trees in our solutions for $h = 4, 5, 6$.

Example. Figure 2 illustrates the h trees representing all the channel schedules in the $\langle h, 2^h - 1 \rangle$ perfect schedule. The window size of $1 \le i \le 2^h - 1$ is 2^j, where $2^j \le i < 2^{j+1}$.

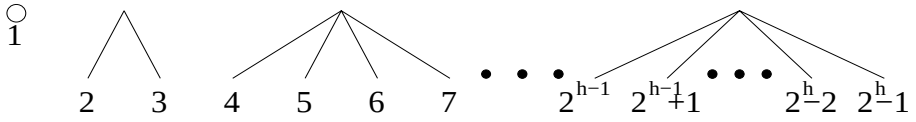


FIG. 2. *The $\langle h, 2^h - 1 \rangle$-schedule.*

3. Asymptotic bounds. Let $W = \langle w_1, w_2, \ldots, w_n \rangle$ be a window vector and let $h(W) = \sum_{i=1}^{n} 1/w_i$. Recall that $H(W)$ is the minimum h such that there exists an $\langle h, W \rangle$-schedule. In subsection 3.1 we show that $\lceil h(W) \rceil \le H(W)$. In subsection 3.4 we show that $H(W) \le h(W) + e \ln(h(W)) + O(1)$. The lower bound comes from a simple observation, while the upper bound comes from the construction of a perfect schedule for W with the required bound on the number of channels. For the upper bound, in subsection 3.2 we present optimal solutions to special cases for the window vector, and then in subsection 3.3 we give an outline of our algorithm and demonstrate it with an example. We conclude this section with subsection 3.5 that shows that these bounds translate to bounds for $N(h)$ for the optimal harmonic windows scheduling problem.

3.1. Lower bound for $H(W)$. We begin with the lower bound.

THEOREM 3. *For any window vector $W = \langle w_1, \ldots, w_n \rangle$,*

$$H(W) \ge \lceil h(W) \rceil .$$

Proof. We need at least h channels where $h-1 < \sum_{i=1}^{n} 1/w_i \le h$. This is because page i requires at least a $1/w_i$ fraction of a channel. □

3.2. Optimal solutions to special window vectors. The upper bound on $H(W)$ is achieved by constructing a perfect schedule. There are a number of steps in the construction, and we start with the simple case where the window sizes are all powers of 2. In this case there is a perfect schedule in which page i is scheduled exactly every w_i slots.

LEMMA 4. *If all the w_i are of the form 2^{v_i} for some $v_i \ge 0$, then there exists a perfect schedule for the windows scheduling problem that uses exactly $H(W) = \lceil h(W) \rceil$ channels, where $\lfloor h(W) \rfloor$ channels are fully utilized.*

Proof. To start, assume that $h(W)$ is an integer and, without loss of generality, that $w_1 \le \cdots \le w_n$. The proof is by induction on n. If $n = 1$, then $w_1 = 1$ and the lemma holds using exactly $h(W) = 1$ channel. If $n > 1$, then there are two cases. In the first case $w_n = 1$, which immediately implies that $w_1 = \cdots = w_n = 1$. Then $n = h(W)$ and the solution dedicates a separate channel to each page. In the second case $w_n > 1$. We claim that $w_n = w_{n-1}$, since otherwise the sum cannot be an integral number because all the $1/w_i$ for $1 \le i \le n-1$ are strictly larger than $1/w_n$. Let W' be the window vector that is constructed from W by deleting w_{n-1} and w_n and adding $w' = w_n/2$. Since $w_{n-1} = w_n$, it follows that $h(W') = h(W)$. By the induction hypothesis on W', we get a solution with $h(W')$ channels in which page $1 \le i \le n-2$ is scheduled every w_i slots and the new page is scheduled every $w_n/2 = w_{n-1}/2$ slots. Furthermore, all $h(W')$ channels are fully utilized. We replace this new page alternately with pages n and $n-1$. In this way these two pages are scheduled perfectly each every $w_n = w_{n-1}$ slots and all the $h(W)$ channels are fully utilized.

Now, suppose that $h(W)$ is not an integer. Because we are assuming powers of 2, the window vector W can be partitioned into two vectors W' and W'' such that $h(W') = \lfloor h(W) \rfloor$ and $h(W'') < 1$. By the proof above W' can be scheduled perfectly in $\lfloor h(W) \rfloor$ fully utilized channels. The window vector W'' can be scheduled in one additional channel as follows. Choose a positive integer $y < 2^{v_n}$ such that $h(W'') + y/2^{v_n} = 1$. Such a y exists since 2^{v_n} is divided by w_i for $1 \le i \le n$. Add y dummy pages each with a window of size 2^{v_n} to the window vector W'' to form a window vector W^* with $h(W^*) = 1$. By the proof above we obtain a perfect schedule for W^* using one channel. To obtain a perfect one channel schedule for W'' we simply omit the dummy pages. □

As a consequence of Lemma 4, we have our first upper bound that is achieved by simply rounding the window sizes down to the nearest power of 2.

LEMMA 5. *For any window vector W, there exists a perfect schedule that uses no more than $\lceil 2h(W) \rceil$ channels.*

Proof. Let $W = \langle w_1, w_2, \ldots, w_n \rangle$. For each w_i let v_i be the nonnegative integer such that $2^{v_i} \le w_i < 2^{v_i+1}$. Let $W' = \langle 2^{v_1}, \ldots, 2^{v_n} \rangle$. Any schedule for W' is also a schedule for W. By Lemma 4, there is a perfect schedule for W' that uses $\lceil h(W') \rceil$ channels. Since $w_i/2 < 2^{v_i}$,

$$h(W') = \sum_{i=1}^{n} \frac{1}{2^{v_i}} < 2 \sum_{i=1}^{n} \frac{1}{w_i} = 2h(W).$$

Thus, at most $\lceil 2h(W) \rceil$ channels are used in the schedule. □

We now improve the upper bound by generalizing Lemma 4. We consider instances in which all the window sizes are powers of 2 multiplied by the same number.

LEMMA 6. *If all the w_i are of the form $u2^{v_i}$ for some $u \geq 1$ and some $v_i \geq 0$, then there exists a perfect schedule for the windows scheduling problem that uses exactly $H(W) = \lceil h(W) \rceil$ channels, where $\lfloor h(W) \rfloor$ channels are fully utilized.*

Proof. Consider the set of windows $w'_i = w_i/u$. By Lemma 4, we can schedule the pages with the w'_i windows on

$$\left\lceil \sum_{i=1}^{n} \frac{1}{w'_i} \right\rceil = \left\lceil u \sum_{i=1}^{n} \frac{1}{w_i} \right\rceil \leq u \lceil h(W) \rceil$$

channels, where

$$\left\lfloor \sum_{i=1}^{n} \frac{1}{w'_i} \right\rfloor = \left\lfloor u \sum_{i=1}^{n} \frac{1}{w_i} \right\rfloor \geq u \lfloor h(W) \rfloor$$

channels are fully utilized. If needed, we add dummy channels with empty schedules to get a schedule with exactly $u \lceil h(W) \rceil$ channels. We now combine sets of u channels together in a round-robin fashion by picking pages alternately from the u channels. In this way the window of each page is multiplied by u. This is fine since by definition $w_i = uw'_i$. Overall, we reduced the number of channels by a factor of u. Hence we used at most $\lceil h(W) \rceil$ channels. When combining the channels, we do it so that the $u \lfloor h(W) \rfloor$ fully utilized channels are combined u at a time. This yields $\lfloor h(W) \rfloor$ fully utilized combined channels. □

3.3. Algorithm outline. Our goal is to construct an algorithm that for a given window vector W creates a perfect schedule with about $h(W) + e \ln(h(W))$ channels (see Lemma 7 for the exact bound). The algorithm is recursive and is based on Lemmas 4 and 6. We first give a high-level description of the algorithm with an illustrative example. The exact details of the algorithm and its analysis appear in subsection 3.4.

An instance of the algorithm is determined by two parameters k and x that are optimized to obtain the best bound. The parameter k represents the depth of the recursion, and the parameter x is optimized for each value of k. First assume that $k = 1$. Here we use Lemma 5 to get a schedule with at most $\lceil 2h(W) \rceil$ channels. Now, fix $k > 1$. Partition the vector of windows into x vectors denoted by W_u for $x \leq u < 2x$. Each window $w_i \in W_u$ will be rounded down to a window w'_i such that $w'_i = 2^{v_i}u'$, where $u = 2^v u'$ for an odd number u'. We round down the window w_i such that w'_i is maximal. The set of all the windows w'_i such that $w_i \in W_u$ is denoted by W'_u. By Lemma 6, we need $\lceil h(W'_u) \rceil$ channels to schedule all the new windows in W'_u. This schedule uses $\lfloor h(W'_u) \rfloor$ channels perfectly with no "waste," while the remaining windows (if they exist) are scheduled on the last channel. This implies a partition of the set W into the perfect vector W^p and the residual vector W^r. Each window $w_i \in W^p$ is scheduled perfectly according to its corresponding w'_i. To schedule the windows in W^r, we recursively apply the algorithm for some $k' < k$.

For each k, we pick the best value for x. We will show that when x is larger, then the w'_i values are closer to the original w_i values. However, the residual set W^r is too big. In order to use the recursion on a smaller residual set, we need a smaller value for x that in turn implies that some of the w'_i are too small compared with their corresponding w_i. This is the tradeoff that dictates the optimization process for the value of x. Finally, we will find the maximum value for k that can be used by this recursive algorithm to get our desired bound.

Example. Consider the following window vector:

$$W = \langle 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 \rangle .$$

It follows that $2 < h(W) < 3$, and therefore we need at least 3 channels to schedule the windows in W. Let us examine first the case $k = 1$ (cf. Lemma 5), in which the new set of windows is

$$W' = \langle 2, 2, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 8, 16, 16, 16, 16 \rangle .$$

The best schedule for W' requires 4 channels. Now assume $k = 2$ and choose $x = 3$. We get the following three vectors W_3, W_4, W_5 and their corresponding three vectors W_3', W_4', W_5':

$$\begin{aligned}
W_3 &= \langle 3, 6, 7, 12, 13, 14, 15 \rangle , \\
W_3' &= \langle 3, 6, 6, 12, 12, 12, 12 \rangle , \\
W_4 &= \langle 2, 4, 8, 9, 16, 17, 18, 19 \rangle , \\
W_4' &= \langle 2, 4, 8, 8, 16, 16, 16, 16 \rangle , \\
W_5 &= \langle 5, 10, 11 \rangle , \\
W_5' &= \langle 5, 10, 10 \rangle .
\end{aligned}$$

For example, the window 7 is rounded down to 6 and therefore belongs to W_3, and the windows $16, 17, 18, 19$ all are rounded down to 16 and therefore belong to W_4. We can use one channel to schedule all the windows in W_3'. We use another channel to schedule the windows $\langle 2, 4, 8, 8 \rangle$ from W_4'. Hence, we get

$$\begin{aligned}
W^p &= \langle 2, 3, 4, 6, 7, 8, 9, 12, 13, 14, 15 \rangle , \\
W^r &= \langle 5, 10, 11, 16, 17, 18, 19 \rangle .
\end{aligned}$$

Now we apply the case $k = 1$ to schedule the windows in W^r. We get

$$W^{r\prime} = \langle 4, 8, 8, 16, 16, 16, 16 \rangle ,$$

which needs only one channel. All together, we need 3 channels: 2 for W^p and 1 for W^r.

3.4. Upper bound for $H(W)$. We are now ready for the major technical lemma which leads to our upper bound. Define r to be a mapping from the positive integers to the reals by

$$r(k) = \begin{cases} 1 & \text{for } k = 1, \\ r(k-1) + \frac{e^{(k-1)/k}}{(e^{(k-1)^2/k} - 1)} & \text{for } k \geq 2. \end{cases}$$

Obviously, the function r is a monotonic increasing function. Furthermore, the $\lim_{k \to \infty} r(k)$ exists and is approximately 4.6412.

LEMMA 7. *For any window vector W and all positive integers k, if $e^{k-1} < h(W) \leq e^k$, then there exists a perfect schedule with the number of channels bounded above by*

$$h(W) + kh(W)^{1/k} + r(k) .$$

Proof. Let $W = \langle w_1, w_2, \ldots, w_n \rangle$. We begin with $k = 1$. By Lemma 5, for any value of $h(W)$, there exists a perfect schedule, denoted by S_1, that uses no more than $\lceil 2h(W) \rceil < 2h(W) + 1 = h(W) + 1h(W)^{1/1} + r(1)$ channels.

Let $k > 1$. Let $e^{k-1} < h(W) \leq e^k$ and assume that the lemma holds for k', $1 \leq k' < k$, using the perfect schedule $S_{k'}$. The schedule S_k for W is defined as follows. Let $x = \lfloor h(W)^{(k-1)/k} \rfloor$. We partition the n windows of window vector W into the x window vectors $W_x, \ldots, W_{2x-1}$ and define a new window size $w'_i \leq w_i$ for each window w_i as follows.

Case $w_i < x$. Let $u = 2^{v_i} w_i$ for v_i such that $x \leq 2^{v_i} w_i < 2x$. Put w_i in W_u and set $w'_i = w_i$.

Case $w_i \geq x$. Let u, $x \leq u < 2x$, be such that $2^{v_i} u \leq w_i < 2^{v_i}(u+1)$ for some $v_i \geq 1$. Put w_i in W_u and set $w'_i = 2^{v_i} u$.

Let W'_u contain those w'_i such that $w_i \in W_u$. This implies a partition into x independent window vectors $W'_x, \ldots, W'_{2x-1}$. All the windows in W'_u have the form $u'2^v$ for an odd u' such that $u = u'2^{v'}$. Hence by Lemma 6, there exists a perfect schedule T_u for W'_u with $\lceil h_u \rceil$ channels, where $h_u = h(W'_u) = \sum_{w'_i \in W'_u} 1/w'_i$. A careful inspection of the proof of this lemma reveals that this schedule uses $\lfloor h_u \rfloor$ channels perfectly with no "waste," where the remaining windows (if they exist) are scheduled on the last channel. This implies a partition of the vector W_u into the perfect vector W^p_u and the residual vector W^r_u. W^p_u contains all the windows $w_i \in W_u$ for which their corresponding pages are scheduled on one of these no-waste channels. W^r_u contains the rest of the windows in W_u. As a result, $h^p_u = \sum_{\{i | w_i \in W^p_u\}} 1/w'_i$ is an integer and $\sum_{\{i | w_i \in W^r_u\}} 1/w'_i < 1$. Since $1/w_i \leq 1/w'_i$, it follows that $h^r_u = \sum_{\{i | w_i \in W^r_u\}} 1/w_i < 1$. Define $W^p = \cup_{x \leq u < 2x} W^p_u$ and $W^r = \cup_{x \leq u < 2x} W^r_u$. The schedule S_k schedules all the windows in W^p using the perfect schedules T_u with exactly $\sum_{u=x}^{2x-1} h^p_u$ channels.

Before describing how to schedule the pages whose windows are in W^r, we analyze the number of channels needed for W^p. For the windows scheduling problem W^p, window sizes $< 2x$ do not change and window sizes $\geq 2x$ are rounded down in each schedule T_u. The worst case happens when $w_i = 2^{v_i}(u+1) - 1$ for $u = x$ and therefore $w'_i = 2^{v_i} x$. This implies that $\frac{w_i}{w'_i} \leq \frac{x+1}{x}$. Hence, the number of channels in the schedule for W^p is

$$\sum_{w_i \in W^p} \frac{1}{w'_i} \leq \frac{x+1}{x} \sum_{w_i \in W^p} \frac{1}{w_i} = \frac{x+1}{x} h(W^p) = \frac{x+1}{x}(h(W) - h(W^r))$$

$$\leq h(W) + \frac{h(W)}{x} - h(W^r)\ . \tag{4}$$

The quantity $h(W)/x$ can be bounded as follows. Since $x = \lfloor h(W)^{(k-1)/k} \rfloor$, $k > 1$, and $e^{k-1} < h(W)$, we have

$$\begin{aligned} h(W)/x &= h(W) / \left\lfloor h(W)^{(k-1)/k} \right\rfloor \\ &\leq h(W)/(h(W)^{(k-1)/k} - 1) \\ &= h(W)^{1/k}/(1 - h(W)^{-(k-1)/k}) \\ &= h(W)^{1/k} \sum_{i=0}^{\infty} (h(W)^{-(k-1)/k})^i \\ &= h(W)^{1/k} + h(W)^{1/k}/(h(W)^{(k-1)/k} - 1) \\ &< h(W)^{1/k} + e^{(k-1)/k}/(e^{(k-1)^2/k} - 1). \end{aligned} \tag{5}$$

The final inequality holds because the function $z/(z^{k-1}-1)$ is decreasing for all $k>1$ and $z>1$.

We are left with the remaining windows, W^r, to be scheduled. Because $h_u^r<1$ for each u and $h(W)\le e^k$, we have $h(W^r)\le e^{k-1}$, as can be seen in the derivation

$$h(W^r)=\sum_{w_i\in W^r}\frac{1}{w_i}=\sum_{u=x}^{2x-1}h_u^r\le x=\left\lfloor h(W)^{(k-1)/k}\right\rfloor\le h(W)^{(k-1)/k} \tag{6}$$

$$\le (e^k)^{(k-1)/k}=e^{k-1}\ . \tag{7}$$

There are two cases to consider depending on whether $h(W^r)\le 1$.

Case 1. $h(W^r)\le 1$. In this case by Lemma 5, W^r has a perfect schedule with the number of channels bounded above by $\lceil 2h(W^r)\rceil\le 2h(W^r)+1$. By this and inequalities (4) and (5), there is a perfect schedule for W that has the number of channels bounded above by

$$h(W)+h(W)^{1/k}+h(W^r)+1+e^{(k-1)/k}/(e^{(k-1)^2/k}-1)\ .$$

Since $k-1\ge 1$ and $h(W)>e^{k-1}$ it follows that $1<(k-1)h(W)^{1/k}$. Since $h(W^r)\le 1$ it follows that $h(W^r)<(k-1)h(W)^{1/k}$. Furthermore, $1+e^{(k-1)/k}/(e^{(k-1)^2/k}-1)\le r(k)$. Hence, we have our bound.

Case 2. $h(W^r)>1$. By inequality (7), we have $e^{k'-1}<h(W^r)\le e^{k'}$ for some $k'<k$. By the induction hypothesis, the perfect schedule $S_{k'}$ for W^r has the number of channels bounded by

$$h(W^r)+k'h(W^r)^{1/k'}+r(k')\ .$$

Claim. $k'h(W^r)^{1/k'}\le (k-1)h(W^r)^{1/(k-1)}$.

Proof of Claim. We show that $az^{1/a}\le bz^{1/b}$ for $1\le a\le b$ for $1\le z\le e^a$. If $a=b$, we are done. If $a<b$, then the function $z^{1/a-1/b}$ is increasing for $z\ge 1$ and has value 1 at $z=1$. Hence, the equation $z^{1/a-1/b}=b/a$ has exactly one solution. Equivalently, the functions $az^{1/a}$ and $bz^{1/b}$ intersect exactly once. Since $az^{1/a}<bz^{1/b}$ for $z=1$, it suffices to show that $az^{1/a}\le bz^{1/b}$ for $z=e^a$. By calculus the function e^y/y in the range $(0,\infty)$ has a minimum at $y=1$. Hence, $e\le e^y/y$ for all $y>0$. Letting $y=a/b$ we have $e\le e^{a/b}/(a/b)$, which is equivalent to $a(e^a)^{1/a}\le b(e^a)^{1/b}$. The claim follows for $a=k'\le b=k-1$ since $h(W^r)\le e^{k'}$.

Because of the above claim and the fact that $r(m)$ is increasing in m, we have that the number of channels for the schedule $S_{k'}$ for W^r is bounded by

$$h(W^r)+(k-1)h(W^r)^{1/(k-1)}+r(k-1)\ .$$

By inequality (6), we have that the number of channels for the schedule $S_{k'}$ for W^r is bounded by

$$h(W^r)+(k-1)h(W)^{1/k}+r(k-1)\ . \tag{8}$$

Combining the bounds (4), (5), and (8), we have that the number of channels for the combined perfect schedules for W^p and W^r together is bounded by

$$h(W)+kh(W)^{1/k}+r(k-1)+e^{(k-1)/k}/(e^{(k-1)^2/k}-1)\ ,$$

which by the definition of r yields our result. □

We are now ready to prove the upper bound on $H(W)$.

THEOREM 8. *Every window vector W, with $h(W) > 1$, has a perfect schedule using the number of channels bounded above by $h(W)+e\ln(h(W))+\eta$, where $\eta < 7.36$.*

Proof. The quantity $\eta = e + \lim_{k\to\infty} r(k)$. Choose $k = \lceil \ln(h(W)) \rceil$. Hence, $e^{k-1} < h(W) \le e^k$. By Lemma 7, there exists a perfect schedule with the number of channels bounded above by $h(W) + kh(W)^{1/k} + r(k)$. This is no more than $h(W) + (\ln(h(W)) + 1)(e^k)^{1/k} + r(k)$, which is equivalent to the claim of the theorem. □

The proof of Theorem 8 implies an approximation algorithm for the optimal windows scheduling problem.

THEOREM 9. *There exists an algorithm for the optimal windows scheduling problem yielding a solution that is within a factor of $1+O(\ln(h(W))/h(W))$ of the optimal solution for any window vector W.*

Proof. Let the algorithm of Theorem 8 produce a schedule with $A(W)$ channels. Recall that $H(W)$ is the optimal number of channels. Without loss of generality we can assume that $h(W) \ge 2$. We have

$$h(W) \le H(W) \le A(W) \le h(W) + e\ln(h(W)) + \eta \ .$$

Hence,

$$\frac{A(W)}{H(W)} \le \frac{h(W) + e\ln(h(W)) + \eta}{h(W)} = 1 + O\left(\frac{\ln(h(W))}{h(W)}\right). \qquad \square$$

3.5. Bounds on $N(h)$. In the optimal harmonic windows scheduling problem the window vectors are restricted to be of the form $W_n = \langle 1, 2, \ldots, n\rangle$. Our lower bound on $H(W_n) = H(n)$ (Theorem 3) leads to the following natural upper bound on $N(h)$.

THEOREM 10.

$$N(h) \le c \cdot e^h$$

for $c = e^{-0.57721} < 0.5615$.

Proof. By Theorem 3, $H(n) \ge h(W_n) = \sum_{i=1}^{n} 1/n$. The quantities $\sum_{i=1}^{n} 1/n$ $(n \ge 1)$ are the well-known harmonic numbers which are known to be bounded below by $\ln(n) + \gamma$, where $\gamma \approx 0.57721...$ is Euler's constant. Hence, $h = H(N(h)) > \ln(N(h)) + \gamma$, which implies the theorem. □

We conclude this section with a lower bound for $N(h)$ that is derived from the upper bound on $H(W)$ from Theorem 8.

THEOREM 11.

$$N(h) \ge \frac{e^h}{e^{\eta+\gamma}h^e} \approx \frac{e^h}{e^{7.93669}h^e} \ .$$

Proof. By Theorem 8, we are looking for n such that $h(W_n)+e\ln(h(W_n))+\eta \le h$. For any such n, $N(h) \ge n$. Since $h(n) = \ln(n) + \gamma$, it follows that for $n = \lfloor \frac{e^h}{e^{\eta+\gamma}h^e} \rfloor$,

$$h(W_n) = h\left(\left\lfloor \frac{e^h}{e^{\eta+\gamma}h^e} \right\rfloor\right) = \ln\left(\left\lfloor \frac{e^h}{e^{\eta+\gamma}h^e} \right\rfloor\right) + \gamma \le h - e\ln(h) - \eta \ .$$

Hence,

$$h(W_n) + e\ln(h(W_n)) + \eta \le h - e\ln(h) - \eta + e\ln(h) + \eta = h \ . \qquad \square$$

With some arithmetic manipulations, one can prove the following corollary.

COROLLARY 12. *For large enough value of h we have that $N(h) \geq (e - \frac{c \ln(h)}{h})^h$ for some constant c.*

Later, in section 5, we will see that the pagoda scheme [15] and our combination technique generalization yield asymptotic results in which $N(h) \geq (e - c')^h$ for some constant c'. Thus, obviously Corollary 12 presents a better asymptotic result.

4. The greedy algorithm. In this section, we describe a greedy strategy for the harmonic windows scheduling problem. It will be clear how this strategy can be generalized to the general windows scheduling problem. However, we do not have analytical bounds for this greedy approach, and we use it only as a heuristic to get better bounds than those produced by our other methods for the harmonic scheduling problem.

The greedy algorithm proceeds in rounds, where in the rth round a perfect placement for page r is found in the channel that minimizes the difference $r - r'$, where r' is the distance between placements of page r. The best placement would be to choose, if possible, the channel in which $r' = r$. This might not be possible; for example, when there are just two channels and we have already perfectly placed 1 in the first channel and 2 in the second channel, then 3 must be placed in the second channel with a distance of 2 apart. In this case $r - r' = 1$. In order to keep track of where placements can be made we represent each channel by a tree, where pages are assigned only to some of the leaves of the tree. If, in some round r, page r cannot be placed perfectly in any tree, then the algorithm terminates having successfully placed $r - 1$ pages.

More formally, let an *open tree* be a tree whose leaves are labeled with two types of labels: pages and windows. A *closed tree* is an open tree that has only page labels. An *open forest* is a collection of open trees, and a *closed forest* is a collection of closed trees. The algorithm maintains an open forest composed of h trees. Initially, all the trees are singleton open trees with window labels whose value is 1. The algorithm terminates when the forest becomes a closed forest. The algorithm runs in rounds. In round r, the algorithm places page r in one of the open trees as follows:

1. Let $w_1 \leq w_2 \leq \cdots \leq w_k$ be the ordered list of the labels of all the leaves in the forest whose labels are of type window.
   (a) Let $m_j = (r \bmod w_j)$ for $1 \leq j \leq k$.
   (b) Let i be an index for which m_i is the minimum among all the m_j. We break ties by selecting the index that is associated with the largest window label.
   (c) Let $d_i = \lfloor r/w_i \rfloor$.
   (d) Let T_s be the tree that contains w_i.
2. If $d_i = 1$, then replace the window label w_i with the page label r in the tree T_s. We call this the *replacement* operation.
3. Otherwise, add d_i children to the leaf associated with w_i replacing this leaf with d_i new leaves. The first child is labeled with the page label r, and the rest are labeled with the window label $w_i \cdot d_i$. We call this the *split* operation.

Note that since $m_i = r - w_i d_i$, we see that our choice of m_i achieves the goal of minimizing $r - r'$, where $r' = w_i d_i$ is the distance between placements of r in the perfect channel schedule that is produced by T_s.

THEOREM 13. *The greedy algorithm constructs a perfect $\langle h, n \rangle$-schedule for some value n.*

Proof. By definition, the algorithm produces h trees that represent h perfect

schedules. The value of n is the last round r in which the forest was still an open forest. It remains to show that indeed the window size of page i is no larger than i. This is true since in both the replacement operation and the split operation, the window size assigned to page i is $w_i d_i$, which is less than r by definition (step (1c)). □

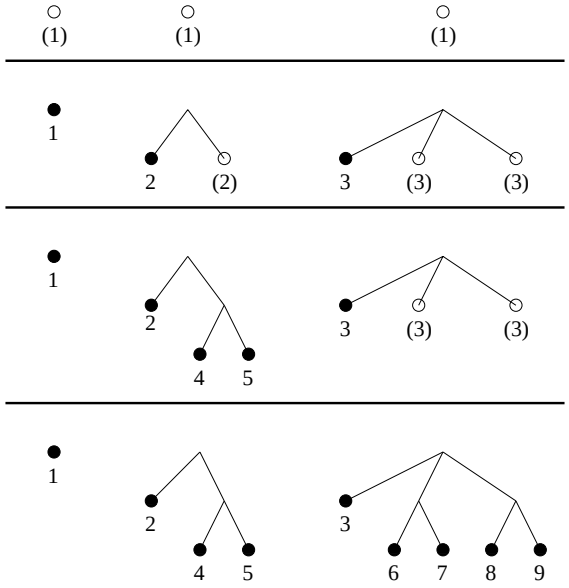


FIG. 3. *The evolution of the greedy algorithm for* $h = 3$.

In Figure 3, we show the evolution of the greedy algorithm for $h = 3$. The final schedule is the perfect $\langle 3, 9 \rangle$-schedule that first appeared in [15]. In the illustration, we use numbers for page labels and numbers in parenthesis for window labels. In addition, we use black circles and white circles to denote leaves with page labels and leaves with window labels, respectively. At the top level, we illustrate the initial open forest that is composed of three singleton open trees whose labels are windows of size 1. In round 1, the label of the first tree becomes the page label 1. This tree is now a closed tree. In round 2, a split operation occurs in the second tree, and it becomes a tree whose root has two children, one with a page label 2 and one with a window label 2. In round 3, a split operation occurs in the third tree, and it becomes a tree whose root has three children, one with a page label 3 and two with a window label 3. Note that since $(3 \bmod 1) < (3 \bmod 2)$, the algorithm selects for the split operation the window label 1 in the third tree rather than the window label 2 in the second tree. The second level of the figure illustrates the three trees after the third round. In round 4, the algorithm splits the node in the second tree with the window label 2 and creates two new leaves: one with a page label 4 and one with a window label 4. This is because $(4 \bmod 2) < (4 \bmod 3)$. Hence, before round 5, the list of available window sizes is $3, 3, 4$. Therefore, the algorithm assigns the page label 5 to the leaf in the second tree with the window label 4. Hereafter, the second tree is closed. The third level of the figure illustrates the three trees after the fifth round. In round 6, the second child of the root of the third tree is split into two leaves: one gets the page label 6 in round 6 and the other gets the page label 7 in round 7. Finally, in round 8, the third child of the root of the third tree is split into two leaves: one gets

the page label 8 in round 8 and the other gets the page label 9 in round 9. At this stage all the trees are closed and the algorithm terminates with the final round $r = 9$, and therefore $n = 9$ as well. The output of the algorithm is illustrated in the fourth level of the figure. Note that pages $2, 3, 4, 6$ get the maximum possible window size, whereas pages $5, 7, 8, 9$ get a smaller than required window size. However, this is the best known solution to the harmonic window scheduling problem for $h = 3$.

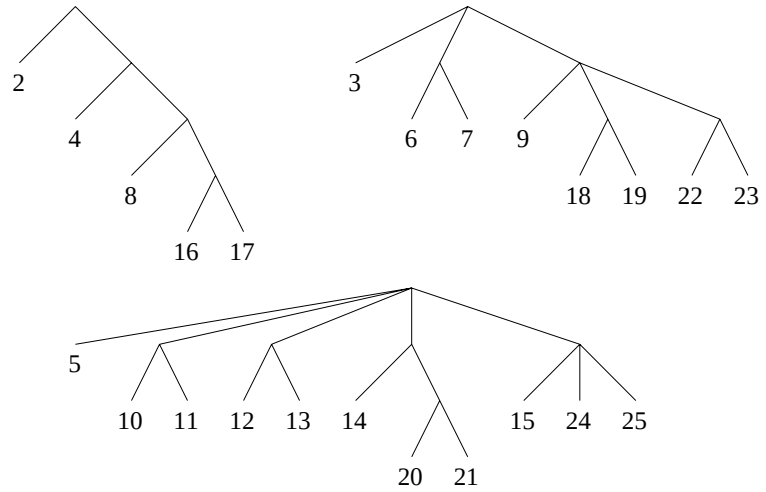


FIG. 4. *The greedy* $\langle 4, 25\rangle$*-schedule.*

Figure 4 illustrates the output of the greedy algorithm for $h = 4$. The greedy algorithm outputs a $\langle 4, 25\rangle$-schedule where there exists a $\langle 4, 26\rangle$-schedule [12]. In particular, this version of the greedy algorithm is not efficient for pages 24 and 25. It could do better by assigning 16 and 17 as siblings of 15 and then assigning 24, 25, and 26 as the children of the sibling of 8.

This leads us to modify the greedy algorithm. We propose two possible modifications. Both try to keep leaves with small window labels open as long as possible.

- The first modification changes the split operation. When d_i is a composite number, it is done in several steps following an increasing order of the prime factors of d_i. For example, if $d_i = 12$ for a node with a window label w_i, then the node is twice split into two nodes and once into three nodes, thus creating five new leaves whose window labels are $2w_i, 4w_i, 12w_i, 12w_i, 12w_i$. One of the window labels $12w_i$ becomes a page label, but the rest remain window labels. In the greedy algorithm that was described earlier, there will be 11 new window labels $12w_i$ and one page label $12w_i$.
- In the second modification, the algorithm follows a different rule in step 1(b) of the algorithm. It sometimes becomes biased towards assigning the new label r to a large window label at the expense of not minimizing $r - r'$. This way, it leaves in the tree smaller window labels for possibly better split operations.

We implemented the greedy algorithm with and without the two modifications above. Unfortunately, it was not the case that one version always outperforms the other versions. However, in most cases the basic greedy algorithm with both modifications was the best. We were able to run our implementations of the basic greedy algorithms for $1 \leq h \leq 20$ and modified versions for $1 \leq h \leq 10$. Table 1 shows

results of the greedy algorithm for $1 \leq h \leq 12$. The best results for $5 \leq h \leq 10$ come from modifications of the basic greedy algorithm. Note that for $h = 12$ the greedy algorithm is within 6% of optimal. In results not shown in the table, for $15 \leq h \leq 20$ the greedy solution is within 3% of optimal.

5. The combination technique. In this section, we develop a technique for solving the optimal harmonic windows scheduling problem. We generalize the technique of [15]. Their pagoda scheme is a special case of the results of this section. We show how to combine schedules together to get new schedules for a larger number of channels. For example, we can combine the $\langle 3, 9\rangle$-schedule with the $\langle 4, 28\rangle$-schedule to produce a $\langle 7, 289\rangle$-schedule (see Theorem 15). However, for $h \leq 20$, the greedy algorithm outperforms any possible combination. Since we have the output of the greedy algorithm only for $h \leq 20$, the combination technique yields our best results for $21 \leq h < H$ for some large integer H. An estimate for H is 1789 and is computed at the end of this section. For $h \geq H$ the asymptotic algorithm of section 3 yields the best bound.

We start with a technical lemma. For $u \leq v$, let an $\langle h, u, v\rangle$-schedule be a schedule on the pages $u, \ldots, v$ on h channels such that page i appears at least once in any consecutive i slots for $u \leq i \leq v$. In particular, an $\langle h, n\rangle$-schedule is an $\langle h, 1, n\rangle$-schedule and an $\langle h-1, 2, n\rangle$ schedule since page 1 gets a dedicated channel. The techniques of this section show how to build new schedules from old ones. As such we define the following notation:

$$\langle h_1, u_1, v_1\rangle, \ldots, \langle h_m, u_m, v_m\rangle \Rightarrow \langle h, u, v\rangle$$

if an $\langle h, u, v\rangle$-schedule can be built from $\langle h_1, u_1, v_1\rangle$- , $\ldots$, $\langle h_m, u_m, v_m\rangle$-schedules. The next lemma shows how to transform an $\langle h, u, v\rangle$ schedule into another schedule for pages from a larger range. We call this the magnification lemma.

LEMMA 14 (magnification lemma). $\langle h, u, v\rangle \Rightarrow \langle h, \ell u, \ell(v+1)-1\rangle$ *for any integer* $\ell \geq 1$.

Proof. Let S be an $\langle h, u, v\rangle$-schedule. We transform S, which is a schedule on the pages $u, \ldots, v$, into another schedule S' on the pages $\ell u, \ldots, \ell(v+1)-1$. We replace page u with pages $\ell u, \ldots, \ell(u+1)-1$ by "stretching" the schedule by a factor of ℓ. In general, we replace the appearances of page $u+j$ in S with the pages $\ell(u+j), \ldots, \ell(u+j+1)-1$ for $0 \leq j \leq v-u$. Since in S the window size of $u+j$ is at most $u+j$, it follows that each of the pages $\ell(u+j), \ldots, \ell(u+j+1)-1$ is guaranteed to have a window of size $\ell(u+j)$. Thus the scheduling of these pages is as required. The smallest page that replaces an appearance of u is ℓu, and the largest page that replaces an appearance of v is $\ell(u+(v-u+1))-1 = \ell(v+1)-1$. This gives us an $\langle h, \ell u, \ell(v+1)-1\rangle$-schedule. □

As an example of Lemma 14, we magnify the $\langle 2, 1, 3\rangle$-schedule

$$\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots \\ 2 & 3 & 2 & 3 & \cdots \end{bmatrix}.$$

Choosing $\ell = 10$, we can create the $\langle 2, 10, 39\rangle$-schedule

$$\begin{bmatrix} 10 & \cdots & 19 & 10 & \cdots & 19 & 10 & \cdots & 19 & 10 & \cdots & 19 & \cdots \\ 20 & \cdots & 29 & 30 & \cdots & 39 & 20 & \cdots & 29 & 30 & \cdots & 39 & \cdots \end{bmatrix}.$$

The above lemma implies our main combination theorem. Here we use $\langle h, n\rangle$ as an abbreviation for $\langle h, 1, n\rangle$.

THEOREM 15 (combination theorem).

$$\langle h_1, n_1\rangle, \langle h_2, n_2\rangle \Rightarrow \langle h_1 + h_2, (n_1+1)(n_2+1)-1\rangle.$$

Proof. The $\langle h_2, n_2\rangle$-schedule is by definition an $\langle h_2, 1, n_2\rangle$-schedule. By Lemma 14, for $\ell = n_1 + 1$, there exists an $\langle h_2, n_1+1, (n_1+1)(n_2+1)-1\rangle$-schedule. The schedule with $h_1 + h_2$ channels is obtained by scheduling pages $1, \ldots, n_1$ on the first h_1 channels following the $\langle h_1, n_1\rangle$-schedule and scheduling pages $n_1+1, \ldots, (n_1+1)(n_2+1)-1$ on the last h_2 channels. ∎

From our examples in the introduction of $\langle 3, 9\rangle$ and $\langle 2, 3\rangle$ we can construct a $\langle 5, 39\rangle$. The first three channels consist of the $\langle 3, 1, 9\rangle$-schedule and the last two channels consist of the $\langle 2, 10, 39\rangle$-schedule constructed above as an example of Lemma 14.

COROLLARY 16. $\langle h, n\rangle \Rightarrow \langle h+1, 2n+1\rangle$.

Proof. Apply Theorem 15 using $\langle 1, 1\rangle$ and $\langle h, n\rangle$. ∎

The above corollary is simple but serves as the base in constructing the folklore $\langle h, 2^h - 1\rangle$-schedule. This schedule is obtained by applying $h-1$ times Corollary 16 starting with the $\langle 1, 1\rangle$-schedule. A better asymptotic result than the $\langle h, 2^h - 1\rangle$-schedule may be obtained by taking a known schedule on $h > 1$ channels and applying the combination theorem, Theorem 15, again and again as in the next theorem.

THEOREM 17. $\langle k, n\rangle \Rightarrow \langle \ell k, (n+1)^\ell - 1\rangle$ *for any integer* $\ell \geq 1$.

Proof. We prove the theorem by induction on ℓ. For $\ell = 1$ the theorem follows trivially. For $\ell > 1$, assume the claim holds for $\ell - 1$. That is, there exists an $\langle (\ell-1)k, (n+1)^{\ell-1} - 1\rangle$-schedule. Then by Theorem 15, using this schedule and the $\langle k, n\rangle$-schedule, there exists an $\langle (\ell-1)k + k, ((n+1)^{\ell-1} - 1 + 1)(n+1) - 1\rangle = \langle \ell k, (n+1)^\ell - 1\rangle$-schedule. ∎

The following corollary is equivalent to Theorem 17.

COROLLARY 18. $\langle k, n\rangle \Rightarrow \langle h, \left(\sqrt[k]{n+1}\right)^h - 1\rangle$ *for* $h = \ell k$.

Corollary 18 implies the existence of an $\langle h, \left(\sqrt[3]{10}\right)^h\rangle$-schedule, provided that 3 divides h, and the existence of an $\langle h, \left(\sqrt[4]{29}\right)^h\rangle$-schedule, provided that 4 divides h. The former is based on the $\langle 3, 9\rangle$-schedule and the latter on the $\langle 4, 28\rangle$-schedule. Both are better than the $\langle h, 2^h - 1\rangle$-schedule but asymptotically far from a potential $\langle h, 0.56147e^h\rangle$-schedule that is implied by Theorem 10. A slightly better asymptotic result than the one in Theorem 17 can be obtained by using the fact that an $\langle h, n\rangle$-schedule is also an $\langle h-1, 2, n\rangle$-schedule.

THEOREM 19. *For an odd* n, $\langle k, n\rangle \Rightarrow \langle \ell(k-1)+1, 2((n+1)/2)^\ell - 1\rangle$ *for any* $\ell \geq 1$.

Proof. We prove the theorem by induction on ℓ. For $\ell = 1$ the theorem follows since $\ell(k-1)+1 = k$ and $2((n+1)/2)^\ell - 1 = n$. For $\ell > 1$ assume the claim holds for $\ell - 1$. That is, there exists an $\langle (\ell-1)(k-1)+1, 2((n+1)/2)^{\ell-1} - 1\rangle$-schedule. We have a $\langle k, n\rangle \Rightarrow \langle k-1, 2, n\rangle$-schedule by deleting the first channel. By Lemma 14 using the factor $((n+1)/2)^{\ell-1}$, we get $\langle k-1, 2, n\rangle \Rightarrow \langle k-1, 2((n+1)/2)^{\ell-1}, 2((n+1)/2)^\ell - 1\rangle$. The claim of the theorem follows by adding the $k-1$ channels of the $\langle k-1, 2((n+1)/2)^{\ell-1}, 2((n+1)/2)^\ell - 1\rangle$-schedule on the pages $2((n+1)/2)^{\ell-1}, \ldots, 2((n+1)/2)^\ell - 1$ to the $(\ell-1)(k-1)+1$ channels of the $\langle (\ell-1)(k-1)+1, 2((n+1)/2)^{\ell-1} - 1\rangle$-schedule on pages $1, \ldots, 2((n+1)/2)^{\ell-1} - 1$ to obtain an $\langle \ell(k-1)+1, 2((n+1)/2)^\ell - 1\rangle$-schedule. ∎

The following corollary is equivalent to Theorem 19.

COROLLARY 20. *For an odd* n, $\langle k, n\rangle \Rightarrow \langle h, 2(\sqrt[k-1]{(n+1)/2})^{h-1} - 1\rangle$ *for* $h =$

$\ell(k-1)+1$. *Equivalently,* $\langle k,n\rangle \Rightarrow \langle h, c(\sqrt[k-1]{(n+1)/2})^h - 1\rangle$ *for* $h = \ell(k-1)+1$ *and* $c = 2/\sqrt[k-1]{(n+1)/2}$.

By the preceding corollary, the $\langle 3,9\rangle$-schedule implies an approximately $\langle h, 0.894427\,(2.23607)^h\rangle$-schedule, provided that 2 divides $h-1$ (the pagoda scheme of [15]), and the $\langle 4,27\rangle$-schedule implies an approximately $\langle h, 0.829827\,(2.41014)^h\rangle$-schedule, provided that 3 divides $h-1$. Note that we cannot use the $\langle 4,28\rangle$-schedule since in this case n is even.

Comparing the asymptotic results. We implemented the greedy algorithm for $1 \le h \le 20$, yielding in particular for $h = 20$ a $\langle 20, 265133521\rangle$-schedule. This in turn implies an asymptotic $\langle h, 0.747366(2.67606)^h\rangle$-schedule, provided that 19 divides $h-1$ (Corollary 20). This value is "close" to the upper bound of $0.56147e^h$ on $N(h)$ (Theorem 10). Using *Mathematica* [18], we get that this schedule is smaller than the $e^h/(e^{7.93669}h^e)$ bound (Theorem 11) for $h > 1789$. That is, the combination technique yields our best results for h in the range [21..1789]. However, implementing the greedy algorithm for $h > 20$ would increase the 1789 threshold. On the other hand, implementing the asymptotic algorithm for $W = \langle 1,\ldots,n\rangle$ would definitely imply a better bound than the one from Theorem 11 and hence would decrease the 1789 threshold. This is because the bounds in section 3 are mainly for the general windows scheduling problem.

6. Solutions for small *h*. In this section we describe our best constructions to the harmonic windows scheduling problem for small h. For $h = 2$ the general $\langle h, 2^h - 1\rangle$-schedule shown in Figure 2 yields a $\langle 2,3\rangle$-schedule, which is optimal. For $h = 3$ the greedy algorithm example shown in Figure 3 yields a $\langle 3,9\rangle$-schedule, which is not known to be optimal. The best possible would be a $\langle 3,10\rangle$-schedule. For $h = 4,5$, we illustrate the $\langle 4,28\rangle$ and the $\langle 5,77\rangle$ nonperfect schedules using the tree representation. We omit the details of our tree representation for the $\langle 6,211\rangle$ nonperfect schedule.

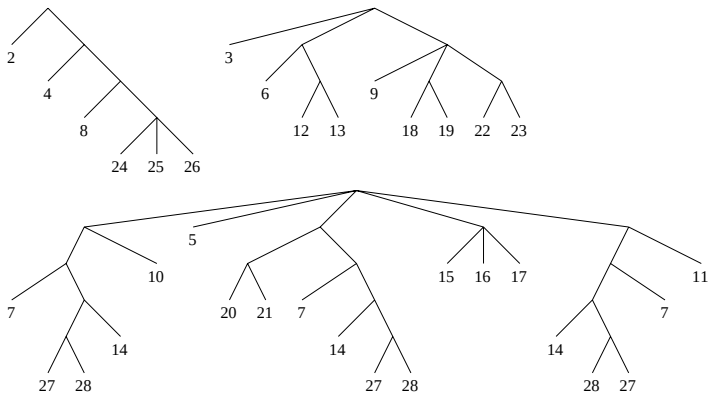


FIG. 5. *The* $\langle 4,28\rangle$*-schedule.*

Our $\langle 4,28\rangle$-schedule is not a perfect schedule. The first channel is dedicated to page 1. Figure 5 uses the tree representation to illustrate the schedules of the second, third, and fourth channels. Channels 1, 2, and 3 have perfect channel schedules but channel 4 does not. This is because pages $7, 14, 27, 28$ occur at multiple leaves of tree 4. The window sizes for page 7 are $7, 7, 6$ repeatedly, for page 14 they are

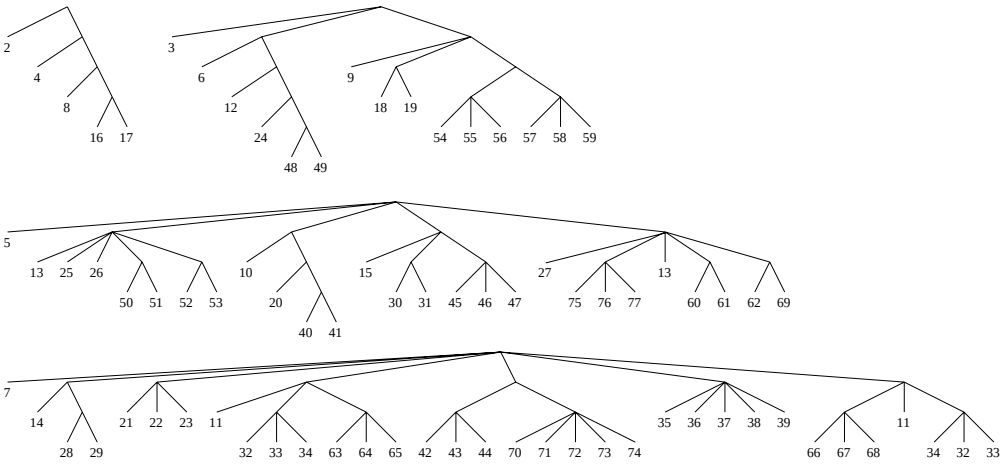


FIG. 6. *The* $\langle 5, 77\rangle$*-schedule.*

$14, 13, 13$ repeatedly, and for pages $27, 28$ they are $27, 27, 26$ repeatedly.

Figure 6 illustrates the four trees representing the second, third, fourth, and fifth schedules in the nonperfect $\langle 5, 77\rangle$-schedule. The first two trees represent perfect schedules; in the other two trees pages $11, 13, 32, 33, 34$ do not have a fixed window size.

7. Open problems. In this section, we discuss some open problems related to the windows scheduling problem, to the harmonic windows scheduling problem, and to the two motivating applications, the push systems with guaranteed delay and the new media-on-demand systems.

The harmonic windows scheduling problem.

- Is the $\langle 3, 9\rangle$-schedule optimal? We conjecture that the answer is yes. However, to show this, more complicated arguments are needed to improve the upper bound of $N(3) \leq 10$ to be $N(3) \leq 9$. These arguments should rely on the *gcd* relationship among the numbers 2 to 9. However, we are able to show that for perfect schedules $N(3) \leq 9$.
- For the media-on-demand application efficient solutions for small values of h are important. Are there better schedules for $h = 4, 5, 6$? The gap for $h = 4$ is $28 \leq N(4) \leq 30$, for $h = 5$ it is $77 \leq N(5) \leq 82$, and for $h = 6$ it is $211 \leq N(5) \leq 226$. For $7 \leq h \leq 20$, our best solutions are outputs of the greedy algorithm. Is there a better algorithm?
- Is there any analytic bound for our greedy algorithm?
- The combination technique yields our best results for $21 \leq h \leq 1789$. A further improvement would be obtained by looking for the best combination for any h using a dynamic programming implementation of the combination theorem.
- We believe that for the optimal harmonic windows scheduling we can get a better lower bound than that given in Theorem 11. This is because we know the exact behavior of the algorithm described in the proof of Lemma 7 in the first stage of the recursion for any choice of x. In the work in progress, we plan to implement the asymptotic algorithm and will probably get better bounds than those implied by the combination technique for $h \leq 1789$.

- Is there another new technique that can yield a provably better asymptotic algorithm for the harmonic windows scheduling problem?

The optimal windows scheduling problem.

- Our asymptotic approximation uses $h(W) + e\ln(h(W)) + O(1)$ channels. Is there an $h(W) + O(1)$ solution, or can that be shown to be impossible?
- Unlike the special case of the optimal harmonic scheduling problem, it is not clear how to define the dual problem of the general windows scheduling problem. One needs a notion of "throughput" to evaluate the performance of a solution for a given number of channels that cannot accommodate all the requested windows.

The media-on-demand and the push systems applications.

- Apparently, the windows scheduling problem does not require a fixed window size or that a page appear only on one channel. Combinatorially, adding these two restrictions separately or combined could be very interesting. Moreover, one could imagine a media-on-demand model in which these restrictions could be useful. In this paper, we have two nonperfect schedules for the small values of $h = 4, 5$. Still, each page is scheduled only on one channel. On one hand, we conjecture that without these restrictions better solutions can be found. On the other hand, we conjecture that asymptotically these restrictions do not reduce the value of $N(h)$.
- Consider the case in which the receiving bandwidth of the clients is less than h. Capitalizing on the work of [16] and with the help of this paper's results we can improve the tradeoff results between $N(r, h)$ and r/h, where $N(r, h)$ is the maximum number of pages that can be scheduled on h channels, provided that the receiving bandwidth of clients is $r < h$. This is a work in progress. The main idea is that having good solutions for $N(r)$ can improve the results for $N(r, h)$.
- In the push systems application, if the servers have a fixed number of channels, there is a need to devise a strategy for "selling" windows. (This is related to the last item of the previous paragraph.)

Acknowledgments. Thanks to Dan Hoke for implementing the greedy algorithm and helping to calculate some of the values in Table 1 and the result of the greedy algorithm for $h \leq 20$. Thanks to Tami Tamir, who carefully read the paper and provided suggestions for improvements. Finally, thanks to the two anonymous referees, whose suggestions improved the presentation of the paper.

REFERENCES

[1] S. Acharya, M. J. Franklin, and S. Zdonik, *Dissemination-based data delivery using broadcast disks*, IEEE Personal Comm., 2 (1995), pp. 50–60.

[2] M. H. Ammar and J. W. Wong, *The design of Teletext broadcast cycles*, Performance Evaluation, 5 (1985), pp. 235–242.

[3] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber, *Minimizing service and operation costs of periodic scheduling*, in Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '98), ACM, New York, 1998, pp. 11–20.

[4] A. Bar-Noy, J. Naor, and B. Schieber, *Pushing dependent data in clients-providers-servers systems*, in Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM '00), 2000, pp. 222–230.

[5] A. Bar-Noy, A. Nisgav, and B. Patt-Shamir, *Nearly optimal perfectly-periodic schedules*, in Proceedings of the 20th ACM Symposium on Principles of Distributed Computing (PODC '01), ACM, New York, 2001, pp. 107–116.

[6] L. Engebretsen and M. Sudan, *Harmonic broadcasting is optimal*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '02), SIAM, Philadelphia, 2002, pp. 431–432.

[7] V. Gondhalekar, R. Jain, and J. Werth, *Scheduling on Airdisks: Efficient Access to Personalized Information Services via Periodic Wireless Data Broadcast*, Technical Report TR-96-25, Department of Computer Science, University of Texas, Austin, TX, 1996.

[8] K. A. Hua, Y. Cai, and S. Sheu, *Exploiting client bandwidth for more efficient video broadcast*, in Proceedings of the 7th International Conference on Computer Communication and Networks (ICCCN '98), 1998, pp. 848–856.

[9] K. A. Hua and S. Sheu, *Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems*, in Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM, New York, 1997, pp. 89–100.

[10] L. Juhn and L. Tseng, *Harmonic broadcasting for video-on-demand service*, IEEE Trans. Broadcasting, 43 (1997), pp. 268–271.

[11] L. Juhn and L. Tseng, *Fast data broadcasting and receiving scheme for popular video service*, IEEE Trans. Broadcasting, 44 (1998), pp. 100–105.

[12] J. Pâris, *A simple low-bandwidth broadcasting protocol for video-on-demand*, in Proceedings of the 8th International Conference on Computer Communications and Networks (IC3N '99), 1999, pp. 118–123.

[13] J. Pâris, S. W. Carter, and D. D. E. Long, *Efficient broadcasting protocols for video on demand*, in Proceedings of the 6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '98), 1998, pp. 127–132.

[14] J. Pâris, S. W. Carter, and D. D. E. Long, *A low bandwidth broadcasting protocol for video on demand*, in Proceedings of the 7th International Conference on Computer Communications and Networks (IC3N '98), 1998, pp. 690–697.

[15] J. Pâris, S. W. Carter, and D. D. E. Long, *A hybrid broadcasting protocol for video on demand*, in Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN '99), 1999, pp. 317–326.

[16] J. Pâris and D. D. E. Long, *Limiting the receiving bandwidth of broadcasting protocols for video-on-demand*, in Proceedings of the Euromedia Conference, 2000, pp. 107–111.

[17] S. Viswanathan and T. Imielinski, *Metropolitan area video-on-demand service using pyramid broadcasting*, ACM Multimedia Systems J., 4 (1996), pp. 197–208.

[18] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computers*, Addison-Wesley, Reading, MA, 1991.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1115–1127

A POLYNOMIAL-TIME ALGORITHM FOR NEAR-PERFECT PHYLOGENY*

DAVID FERNÁNDEZ-BACA† AND JENS LAGERGREN‡

Abstract. A parameterized version of the Steiner tree problem in phylogeny is defined, where the parameter measures the amount by which a phylogeny differs from "perfection." This problem is shown to be solvable in polynomial time for any fixed value of the parameter.

Key words. algorithms, computational biology, character-based methods, evolutionary trees, parsimony, perfect phylogeny, phylogeny, Steiner tree

AMS subject classifications. 68Q25, 68R05, 68R10, 68W40, 92B99

DOI. 10.1137/S0097539799350839

1. Introduction.

A fundamental problem in biology and linguistics is that of inferring the evolutionary history of a set of taxa, each of which is specified by the set of *traits* or *characters* that it exhibits [4, 6, 15]. Formally, let C be a set of *characters*, and for every $c \in C$ let $\mathcal{A}_c$ be the set of allowable *states* for character c. Let $m = |C|$ and $r_c = |\mathcal{A}_c|$. A *species* s is an element of $\mathcal{A}_1 \times \cdots \times \mathcal{A}_m$; $c(s)$ is referred to as the *state of character* c *for* s. A *phylogeny* for a set of n distinct species S is tree T with the following properties:

(C1) $S \subseteq V(T) \subseteq \mathcal{A}_1 \times \cdots \times \mathcal{A}_m$,

(C2) every leaf in T is in S.

Define the *length* of a phylogeny T for S as

$$\text{length}(T) = \sum_{(u,v)\in E(T)} \text{dist}(u, v),$$

where, for any two species u, v, $\text{dist}(u, v)$ denotes the number of character states in which u and v differ (that is, $\text{dist}(u, v)$ is the Hamming distance between u and v). The *Steiner tree problem in phylogeny* (STP) is to find a phylogeny T of minimum length for a given set of species S.

STP and many of its variants are known to be NP-hard [7, 3]. While polynomial-time approximation algorithms with constant ratio bound are known for this problem (for a recent example, see [11]), there are limits to the approximability of STP [5].

STP is related to the problem of determining whether S has a *perfect* phylogeny, i.e., one that satisfies (C1), (C2), and the following:

(C3) For every $c \in C$ and every $\sigma \in \mathcal{A}_c$, the set of all $u \in V(T)$ such that $c(u) = \sigma$ induces a subtree of T.

*Received by the editors January 27, 1999; accepted for publication (in revised form) March 13, 2003; published electronically August 6, 2003. A preliminary version of this paper was presented at the 23rd International Conference on Automata, Languages, and Programming, Paderborn, Germany, 1996.

http://www.siam.org/journals/sicomp/32-5/35083.html

†Department of Computer Science, Iowa State University, Ames, IA 50011-1041 (fernande@cs.iastate.edu). The work of this author was supported in part by the National Science Foundation under grants CCR-9211262, CCR-9520946, and CCR-9988348.

‡Department of Numerical Analysis and Computer Science, Royal Institute of Technology, Stockholm, Sweden (jensl@nada.kth.se). The work of this author was supported by grants from the NFR and TFR.

The perfect phylogeny problem was shown to be NP-complete by Bodlaender, Fellows, and Warnow [2] and, independently, by Steel [14]. This motivated the study of the fixed-parameter versions of the problem, where either m or r is fixed. Both versions have been shown to be polynomially solvable, the first by McMorris, Warnow, and Wimer [13], and the second by Agarwala and Fernández-Baca [1]. The time bound of the latter's algorithm was improved by Kannan and Warnow [12].

If a set of species admits a perfect phylogeny, the underlying set of characters C is *compatible*; thus, the perfect phylogeny problem is often called the *character compatibility problem*. In practice most sets of characters are incompatible, and thus a natural problem is to find a maximum-cardinality subset of C that is compatible. This problem is, unfortunately, equivalent to CLIQUE [8] and hence not only NP-hard, but also extremely hard to solve approximately [10].

The difference between m and the maximum-cardinality compatible subset of C is one measure of the degree of compatibility of a set of species. Here we study a measure of incompatibility that we believe is equally natural, which is motivated by the following result [1, 9].

THEOREM 1. *Let T^* be a phylogeny for S. Then* $\text{length}(T^*) \geq \sum_{c \in C}(r_c - 1)$ *and T^* is a perfect phylogeny if and only if* $\text{length}(T^*) = \sum_{c \in C}(r_c - 1)$.

Thus, the length of a perfect phylogeny (assuming one exists) gives a tight lower bound on the length of any phylogeny for S. Motivated by this observation, let us define the *penalty* of a phylogeny T as

$$\text{penalty}(T) = \text{length}(T) - \sum_{c \in C}(r_c - 1).$$

Obviously, STP can be rephrased as the problem of finding a phylogeny T such that $\text{penalty}(T)$ is minimum. We are interested in the fixed-parameter version of the problem, namely, given a set of species S and an integer q, does S have a phylogeny with penalty at most q? We show that for each fixed q and r, the resulting "near-perfect" phylogeny problem can be solved in polynomial time. The running time of our algorithm is a polynomial whose degree depends on the parameters, making the algorithm practical only for small values of the parameters. On the other hand, the flexibility of allowing one or more characters to violate condition (C3) by some fixed amount may extend the range of applicability of character-based methods.

Our near-perfect phylogeny algorithm shares several ideas with earlier work on the perfect phylogeny problem [1, 12]. As in the algorithms for the latter problem, we rely on the observation that there is a polynomially bounded number of ways in which species can be partitioned into subfamilies that respect state boundaries for some character. (See section 2 for a precise definition.) The approach is to build subphylogenies for these subfamilies, proceeding by increasing cardinality. Subphylogenies are joined through their roots to form subphylogenies for larger subfamilies.

The construction of subphylogenies is complicated by issues that do not arise in the perfect phylogeny problem. Each edge in a perfect phylogeny corresponds to a *character partition*, i.e., a partition of S into subfamilies such that there exists a character c on which no state is shared between species of different subfamilies. This property and the fact that the number of character partitions is polynomially bounded when r is fixed are keys to the efficient solution of the perfect phylogeny problem. Unfortunately, it can easily be seen that imperfect phylogenies may have bad edges, i.e., edges not inducing character partitions. We show, however, that the number of bad edges is polynomially bounded when the penalty is bounded.

Our strategy to build a subphylogeny for a subset of species is therefore to generate different candidate trees consisting only of bad edges and use them as skeletons from which to hang subphylogenies for character subfamilies. From among all of the trees thus enumerated, we select the one that results in the least penalty. It is nontrivial to determine which subphylogenies to connect to a candidate bad tree, because there is no a priori bound on the degree of a vertex. (Such a bound would imply that the subphylogenies could be found in polynomial time by simply trying all combinations of character subfamilies.) We show that it suffices to enumerate a polynomially bounded number of labeled candidate trees.

The rest of the paper is organized as follows. Section 2 gives definitions and notation. Section 3 explains how to compute perfect phylogenies. The properties of low-penalty phylogenies and subphylogenies are studied in section 4. In particular, bounds are derived there on the number of bad edges in a near-perfect phylogeny and on the amount of information that must be enumerated to construct such a phylogeny. Our near-perfect phylogeny algorithm is presented in section 5. Section 6 concludes the paper.

2. Basic definitions and notation. The vertex sets of all trees are assumed to be subsets of $\mathcal{A}_1 \times \cdots \times \mathcal{A}_m$. Note that this implies that every two adjacent nodes are distinct. No generality is lost, since a tree that does not satisfy this condition can be transformed into one that does and that has at most the same length.

Throughout the paper, r denotes $\max_{c \in C} r_c$.

Let c be a character. We assume that each state in $\mathcal{A}_c$ is exhibited by some element of S. Obviously, any state that is not exhibited by any species can be deleted from $\mathcal{A}_c$. We assume that $\mathcal{A}_i \cap \mathcal{A}_j = \emptyset$ for $i \neq j$. No generality is lost by making this assumption, since it can always be enforced by replacing each state σ on every character c by the state (σ, c).

Let T be a tree, and let $\sigma \in \mathcal{A}_c$. Then $T[\sigma]$ denotes the subgraph of T induced by all nodes v such that $c(v) = \sigma$. For any edge $(u, v) \in T$, T_u and T_v denote the components of $T - \{(u, v)\}$ containing u and v, respectively.

DEFINITION 1. *Let T be a phylogeny for S. Character c is* convex *in T if for every $\sigma \in \mathcal{A}_c$, $T[\sigma]$ is connected. If $T[\sigma]$ is not connected, σ is a* penalty state *in T and c is* nonconvex *in T.*

In what follows, $C_p \subseteq C$ denotes a set of characters that are required to be convex.

DEFINITION 2. *Let T be a phylogeny for S. T is q-*near-perfect *if* penalty$(T) \leq q$. *T is C_p-*perfect *if every $c \in C_p$ is convex in T. If $C_p = C$, T is simply called* perfect. *A* minimum C_p-perfect phylogeny *is a C_p-perfect phylogeny of minimum length.*

An example of a C_p-perfect phylogeny is shown in Figure 1. For clarity, the set of states for every character is written as $\{0, 1, 2, 3, 4\}$; in reality, $\mathcal{A}_{c_i} = \{(j, c_i)\}_{j=0}^{4}$ for $i = 1, 2, 3, 4$. We shall refer to Figure 1 throughout the paper to illustrate various concepts.

Clearly, a C_p-perfect phylogeny may not exist for a given set C_p. Note also that, by Theorem 1, all perfect phylogenies have the same (minimum) length, so it is redundant to talk about minimum perfect phylogenies.

DEFINITION 3. *Two subsets X, Y of S* share a state *on $c \in C$ if there exists a state σ of c such that $c(x) = c(y) = \sigma$ for some $x \in X$, $y \in Y$. State σ is referred to as a* shared state.

DEFINITION 4. *A* character partition *with respect to a character c is a partition (S_1, S_2) of S such that no species in S_1 shares a state of c with any species of S_2. The subsets S_1 and S_2 are* character subfamilies. *A character subfamily Q is* proper *if at*

	c_1	c_2	c_3	c_4
s_1	0	2	1	2
s_2	4	2	1	1
s_3	0	2	3	1
s_4	0	4	4	0
s_5	0	1	0	0
s_6	3	2	1	0
s_7	2	0	1	3
s_8	1	0	1	2
s_9	0	2	2	4
s_{10}	0	3	2	2

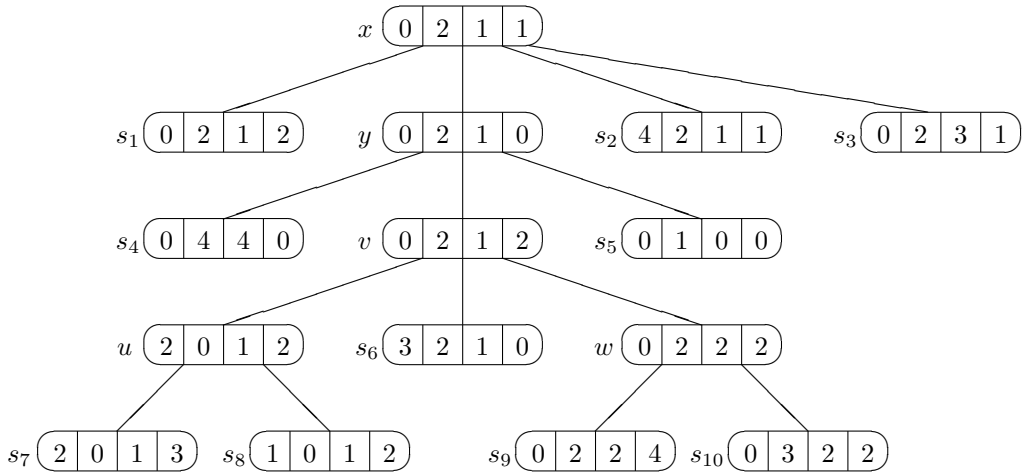


FIG. 1. *Top: A set of ten species described by their states on a set of characters* $C = \{c_1, c_2, c_3, c_4\}$*, each with five states. Bottom: A* C_p*-perfect phylogeny for the set of species, where* $C_p = \{c_1, c_2, c_3\}$*. The length of the tree is* 18*; its penalty is* 2.

most one state is shared between Q *and* $S - Q$ *for every* $c \in C_p$*.*

From this point forward, all character subfamilies that we consider are assumed to be proper.

The number of character subfamilies is $O(2^r m)$, since at most 2^r are defined by the states of any given character [1]. This bound is polynomial when r is fixed, which motivates the following approach to computing minimum C_p-perfect phylogenies: Enumerate the subfamilies by increasing cardinality, and for each subfamily find a minimum-length rooted C_p-perfect phylogeny made up of phylogenies for smaller subfamilies. Since our goal is to compose the phylogenies by linking roots through edges, the permissible states for the roots are partially determined by convexity. To formalize these ideas, we need some definitions. In what follows, $*$ denotes an unspecified state, which is in none of the $\mathcal{A}_i$'s.

DEFINITION 5. *Let* $Q \subseteq S$ *be a character subfamily. The* splitting vector *of* Q *is the species* $Sv(Q)$ *where, for each character* c*, if* $c \in C_p$ *and* Q *and* $S - Q$ *share state* σ *on character* c*, then* $c(Sv(Q)) = \sigma$*; otherwise,* $c(Sv(Q)) = *$*.*

DEFINITION 6. *Let* Q*,* Q_1 *be character subfamilies such that* $Q_1 \subset Q$*.* Q *and* Q_1 *are* compatible *if for every* $c \in C_p$ *such that* $c(Sv(Q)), c(Sv(Q_1)) \neq *$*,* $c(Sv(Q)) =$

$c(Sv(Q_1))$.

Intuitively, if Q and Q_1 are compatible, there conceivably exists a C_p-perfect phylogeny for $Q \cup \{Sv(Q)\}$ such that one of the subtrees of $Sv(Q)$ is a phylogeny for $Q_1 \cup \{Sv(Q_1)\}$. In such a phylogeny, the states of the root on some characters $c \in C_p$ such that $c(Sv(Q)) = *$ may have to take on specific values, because a state on c may be shared between Q_1 and $S - Q_1$ that is not shared between Q and $S - Q$. This motivates the following definition.

DEFINITION 7. *Let Q, Q_1 be compatible character subfamilies such that $Q_1 \subset Q$. The* splitting vector *for (Q, Q_1) is the species $Sv(Q, Q_1)$ where for each character c, if $c \in C_p$ and state σ is shared between Q and $S - Q$ or between Q_1 and $S - Q_1$, then $c(Sv(Q, Q_1)) = \sigma$; otherwise, $c(Sv(Q, Q_1)) = *$.*

DEFINITION 8. *Let $Q \subseteq S$ and x be a species. Then $\sim_x$ denotes the equivalence relation on Q defined as the transitive closure of the following relation R_x: For $s, t \in Q$, $(s, t) \in R_x$ if there exists a character c such that $c(s) = c(t) \neq c(x) \neq *$. Denote by Q/x the collection of equivalence classes of $\sim_x$.*

Observe that each of the sets in Q/x must be in the same connected component of $T - \{x\}$ for any perfect (not just C_p-perfect) phylogeny T for $Q \cup \{x\}$.

DEFINITION 9. *Let T be a phylogeny for S and let $e = (u, v)$ be an edge of T. Then $(S \cap V(T_u), S \cap V(T_v))$ is an* edge partition *of S (with respect to T). The subsets $S \cap V(T_u)$ and $S \cap V(T_v)$ are* edge subfamilies. *Edge (u, v) is* good *if the partition $(S \cap V(T_u), S \cap V(T_v))$ induced by e is a character partition; otherwise, e is* bad.

To close this section, we illustrate some of the concepts introduced here, making reference to Figure 1. Let $Q_1 = \{s_9, s_{10}\}$ and $Q = \{s_7, s_8, s_9, s_{10}\}$. Then, $Sv(Q_1) = (0, 2, *, *)$ and $Sv(Q) = (0, 2, 1, *)$; thus, Q_1 and Q are compatible, and $Sv(Q, Q_1) = (0, 2, 1, *)$. In the phylogeny shown, edges (s_1, x), (x, y), and (y, v) are bad; all other edges are good.

3. Finding perfect phylogenies. Before studying near-perfect phylogenies, we review the perfect phylogeny algorithm of Agarwala and Fernández-Baca and the improvements devised by Kannan and Warnow. The algorithm relies on two facts. The first is the aforementioned polynomial bound (for fixed r) on the number of character subfamilies. The second is that perfect phylogenies can be assembled from phylogenies for character subfamilies, because, as shown in [1], every edge in a perfect phylogeny for S is good.

DEFINITION 10. *Let Q be a character subfamily. A* perfect subphylogeny *for Q is a rooted perfect phylogeny for Q, whose root x satisfies $c(x) = c(Sv(Q))$ for all c such that $c(Sv(Q)) \neq *$, and $c(x) = c(s)$ for some $s \in Q$ otherwise.*

It is straightforward to verify that if Q_1 and $Q_2 = S - Q_1$ have perfect subphylogenies T_1 and T_2, respectively, then the tree obtained by connecting the roots of T_1 and T_2 by an edge is a perfect phylogeny for S.

DEFINITION 11. *Let Q, Q_1 be compatible character subfamilies such that $Q_1 \subset Q$. A* perfect subphylogeny *for (Q, Q_1) is a rooted perfect phylogeny for Q, whose root x is such that* (i) *$c(x) = c(Sv(Q, Q_1))$ for all c such that $c(Sv(Q)) \neq *$, and $c(x) = c(s)$ for some $s \in Q$ otherwise, and* (ii) *the removal of x partitions Q into subsets some of which union to Q_1.*

The following result is proved in [12, 1].

LEMMA 2. *Suppose that Q is a character subfamily and that $Q_1 \subset Q$ has a subphylogeny. Let $Q_2 = Q - Q_1$. Then, (Q, Q_1) has a subphylogeny if and only if* (i) *Q_2 has a subphylogeny or* (ii) *every element of $Q_2/Sv(Q, Q_1)$ has a subphylogeny. In case* (ii)*, $c(Sv(Q, Q_1)) \neq *$ for every character c.*

SUBPHYLOGENY(Q).
For each subfamily $Q_1 \subset Q$ compatible with Q do the following:

1. Let $T_{Q_1} = \mathcal{N}(Q_1)$.
2. If $T_{Q_1} \neq \emptyset$, then do the following:
   (a) Let $Q_2 = Q - Q_1$ and $T_{Q_2} = \mathcal{N}(Q_2)$.
   (b) If $T_{Q_2} \neq \emptyset$, then let T_Q be the subphylogeny for Q whose root is a node x_Q satisfying $c(x_Q) = c(Sv(Q, Q_1))$ for every $c \in C$ such that $c(Sv(Q, Q_1)) \neq *$, and $c(x_Q) = c(x_{Q_1})$ for every other c, where x_{Q_1} is the root of T_{Q_1}. Set $\mathcal{N}(Q) = T_Q$ and return.
   (c) Otherwise, let $\{P_i\}_{i=1}^{k}$ be the set of equivalence classes $Q_2/Sv(Q, Q_1)$. If $T_{P_i} = \mathcal{N}(P_i) \neq \emptyset$ for every $i \in \{1, \ldots, k\}$, then let T_Q be the subphylogeny for Q whose root x_Q satisfies $c(x_Q) = c(Sv(Q, Q_1))$ for every $c \in C$ and whose subtrees are T_{Q_1} and $T_{P_1}, \ldots, T_{P_k}$. Set $\mathcal{N}(Q) = T_Q$ and return.

PERFECT-PHYLOGENY(S, C).

1. Allocate a table $\mathcal{N}$ with one entry for each character subfamily Q. Set $\mathcal{N}(Q) = \emptyset$ for each such Q.
2. Enumerate, by increasing cardinality, each character subfamily Q, and run SUBPHYLOGENY(Q).
3. If there exists a pair of subfamilies Q_1, Q_2 such that $Q_2 = S - Q_1$ and $\mathcal{N}(Q_1), \mathcal{N}(Q_2) \neq \emptyset$, then return the tree T obtained by linking the roots of $T_{Q_1} = \mathcal{N}(Q_1)$ and $T_{Q_2} = \mathcal{N}(Q_2)$ by an edge. Otherwise, return $\emptyset$.

FIG. 2. *The perfect phylogeny algorithm.*

This leads to the algorithm of Figure 2. The main procedure, PERFECT-PHYLOGENY, considers character subfamilies by increasing cardinality; it attempts to build a subphylogeny for each one using procedure SUBPHYLOGENY, inserting the result into a table $\mathcal{N}$.

PERFECT-PHYLOGENY iterates over all $O(2^r m)$ character subfamilies Q. For each of these, SUBPHYLOGENY considers $O(2^r m)$ choices of Q_1. Kannan and Warnow show how to find the equivalence classes of $Q_2/Sv(Q, Q_1)$ in $O(n)$ time at the expense of precomputing, in $O(2^r nm^2)$ time, the equivalence classes of $S/Sv(G)$ for every subfamily G (see [12]). An $O(2^{2r} nm^2)$ bound follows.

4. Near-perfect phylogenies. The algorithm of Figure 2 relies heavily on the fact that perfect phylogenies have no bad edges, a property that may not hold for near-perfect phylogenies. In this section, we show that near-perfect phylogenies can be decomposed into perfect and imperfect parts. The former can be handled by the techniques described in the previous section. We prove that the latter can be generated by examining an amount of information that is polynomial for each fixed q. As before, C_p denotes a set of characters required to be convex. Before proceeding, we need some definitions.

DEFINITION 12. *A* penalty state assignment *is a function α that maps each $c \in C - C_p$ to an element $\alpha(c)$ of $\mathcal{A}_c$. The* penalty state assignment *of a species s is the penalty state assignment α_s, where $\alpha_s(c) = c(s)$ for each $c \in C - C_p$.*

DEFINITION 13. *Let Q be a character subfamily and α be a penalty state assignment. A* subphylogeny *for (Q, α) is a rooted C_p-perfect phylogeny T for Q whose root x satisfies the following properties:*

(i) *For every* $c \in C_p$, $c(x) = c(Sv(Q))$ *if* $c(Sv(Q)) \neq *$; *otherwise,* $c(x) = c(u)$ *for some* $u \in Q$.
(ii) *For every* $c \in C - C_p$, $c(x) = \alpha(c)$.

T *is a* minimum-length subphylogeny *if it has the smallest length among all subphylogenies for* (Q, α).

DEFINITION 14. *Let* T *be a subphylogeny with root* x *for* (Q, α). *Let* (u, v) *be an edge of* T, *where* u *is the parent of* v. *Then,* (u, v) *is* good *if* $S \cap V(T_v)$ *is a character subfamily; otherwise* (u, v) *is* bad. *The maximal subtree* T *that contains* x *and only bad edges is the* bad tree *of* T *and is denoted* $B(T)$. T *is in* normal form *if, for every good edge* (u, v) *in* T *such that* u *is in* $B(T)$, T_v *is a subphylogeny for some pair* (Q_v, α_v), *where* $Q_v \subseteq Q$.

Note that, by the maximality of $B = B(T)$, if an edge of T is not in B but is adjacent to an edge of B, then it is good; i.e., the associated edge subfamily is a character subfamily as well.

DEFINITION 15. *Let* T *be a subphylogeny,* v *be a node of* $B = B(T)$, *and* u *be a child of* v *not in* B. *Then the subset of* S *contained in* T_u *is an* edge subfamily at v. *An edge subfamily* Q *at* v *is* perfect *if no state is shared between* Q *and* $S - Q$ *on a character* c *except (possibly)* $c(v)$, *and* $Q \cup \{v\}$ *has a perfect phylogeny. All other edge subfamilies at* Q *are* imperfect.

For a node v in B, $Pe(v)$ and $Im(v)$ stand for the sets of perfect and imperfect edge subfamilies at v, respectively. $\mathcal{P}(v)$ is the union of all perfect edge subfamilies at v, and $\mathcal{F}(v)$ is the union of all edge subfamilies at v.

Let T be a subphylogeny for (Q, α), and let v be a node in T. Then, by definition, the subtree of T consisting of v, together with all T_u such that $S \cap V(T_u)$ is a perfect edge subfamily at v, is a subphylogeny for $\mathcal{P}(v)$.

We now illustrate some of the notions introduced so far, making reference to Figure 1. The subtree rooted at y in that diagram is a subphylogeny for (Q, α), where $Q = \{s_4, s_5, s_6, s_7, s_8, s_9, s_{10}\}$ and $\alpha(c_4) = 0$; its bad tree consists of edge (y, v). Indeed, if the set $P = \{s_1, \ldots, s_{10}\}$ is a character subfamily within a larger set S such that P and $S - P$ share (say) state $(1, c_3)$, then the whole tree at x is a subphylogeny for (P, α), where $\alpha(c_4) = 1$. The bad tree in this case consists of edges (s_1, x), (x, y), and (y, v). Both of the subphylogenies we have described are in normal form. Observe that $Pe(v) = \{\{s_7, s_8\}, \{s_9, s_{10}\}\}$ and $Im(v) = \{\{s_6\}\}$.

The results that follow characterize the structure of near-perfect phylogenies and subphylogenies.

LEMMA 3. *Let* T *be a* C_p*-perfect phylogeny, and let* (u, v) *be a bad edge in* T. *Then, for each* $c \in C_p$, $c(u) = c(v)$. *Furthermore, there is some* $c \in C - C_p$ *such that* $c(u) \neq c(v)$.

Proof. We first show that for each $c \in C_p$ there exists a shared state on c between $Q_u = S \cap V(T_u)$ and $Q_v = S \cap V(T_v)$. Suppose that this is false. Then (Q_u, Q_v) is a character partition and (u, v) is, by definition, a good edge, a contradiction.

Because of the state shared between Q_u and Q_v on $c \in C_p$ and the fact that T is C_p-perfect, we must have $c(u) = c(v)$. We must have $c(u) \neq c(v)$ for some $c \in C - C_p$ because we are dealing with phylogenies where every two nodes differ in at least one state. □

LEMMA 4. *Let* T *be a* C_p*-perfect phylogeny such that* penalty$(T) \leq q$. *Then* T *has at most* qr *bad edges.*

Proof. Let $C' = C - C_p$. For each $c \in C'$ let l_c be the number of edges (u, v) in T such that $c(u) \neq c(v)$. Since penalty$(T) \leq q$, $|C'| \leq q$. By Lemma 3, for every

bad edge (u, v) there must be some $c \in C'$ such that $c(u) \neq c(v)$. Thus, the number of bad edges is at most $\sum_{c \in C'} l_c$. Moreover, $\sum_{c \in C'}(l_c - (r_c - 1)) \leq q$. Hence, the number of bad edges is bounded by $q + \sum_{c \in C'}(r_c - 1) \leq qr$. ∎

LEMMA 5. *Suppose that Q is a character subfamily having a rooted C_p-perfect phylogeny T, with root x such that, for every $c \in C_p$, $c(x) = c(Sv(Q))$ if $c(Sv(Q)) \neq *$. Let α be the penalty state assignment of x. Then, (Q, α) has a subphylogeny of length at most* length(T).

Proof. T is a subphylogeny for (Q, α), except that there might be some $c \in C_p$ such that $c(x) \neq c(u)$ for any $u \in Q$. For each such c, carry out the following step until it no longer applies:

> Let $c(x) = \sigma$ be such that $\sigma \neq c(u)$ for any $u \in Q$. Let A be the connected component of $T[\sigma]$ containing x, and let (u, v) be any edge of T such that $u \in A$ and $v \notin A$. Then, $c(u) = \beta \neq \sigma$. Set $c(w)$ equal to β for all w in A.

Each application of the above step preserves perfection with respect to C_p; furthermore, it does not affect the contribution of the nonconvex characters to the length. When the step no longer applies, T is a subphylogeny. ∎

LEMMA 6. *Suppose that the pair (Q, α) has a subphylogeny. Then, (Q, α) has a minimum-length subphylogeny in normal form.*

Proof. Let T be a minimum-length subphylogeny for (Q, α), and let $B = B(T)$. If T is in normal form, we are done, so suppose it is not. Successively consider each good edge (u, v) in T such that u is in B and T_v is not a subphylogeny for (Q_v, α_v), where $Q_v = S \cap V(T_v)$ and α_v is the penalty state assignment of v. For each such edge, apply the following transformation to T: Let T'_v be a subphylogeny for (Q_v, α_v) such that length$(T'_v) \leq$ length(T_v); such a tree T'_v exists by Lemma 5, since T_v is a C_p-perfect phylogeny for Q_v where, for every $c \in C_p$, $c(v) = c(Sv(Q_v))$ if $c(Sv(Q_v)) \neq *$. Replace T_v by T'_v by deleting T_v and making the root of T'_v a child of u.

Each application of the transformation preserves the properties that T is of minimum length and that T is a subphylogeny for (Q, α). After the final application, T is in normal form. ∎

LEMMA 7. *Let T be a subphylogeny for (Q, α), let $B = B(T)$, and let*

$$U = (S - Q) \cup \{u \in P : P \in Im(v), v \in B\}.$$

Then, for each $v \in V(B)$, $\mathcal{P}(v) = G(v) - U$ for some set $G(v) = \bigcap_{i=1}^{l} Q_i$, where $\{Q_i\}_{i=1}^{l}$ is a set of character subfamilies for different characters $c \in C - C_p$.

Proof. Pick $G(v)$ as follows. For each node v of B and each nonconvex character c, let $Q(v, c)$ be the character subfamily consisting of all species $s \in S$ such that $c(s) = c(x)$ for some $x \in \mathcal{P}(v)$. The set $G(v)$ is the intersection of $Q(v, c)$ over all characters $c \in C - C_p$.

To prove the claim, we show containment in both directions:

- Suppose $s \in \mathcal{P}(v)$. Then, $s \in Q(v, c)$ for each $c \in C - C_p$. Hence, $s \in G(v) - U$.
- Suppose $s \in G(v) - U$. By definition of $G(v)$, for each $c \in C - C_p$ there is a species $x \in \mathcal{P}(v)$ such that $c(x) = c(s)$. Also, there must exist a node $u \in B$ such that $s \in \mathcal{F}(u)$. Since $s \notin U$, $c(s) = c(v) = c(u)$ for every $c \in C - C_p$. But then, by Lemma 3 we must have $u = v$. Hence, $s \in \mathcal{P}(v)$. ∎

LEMMA 8. *Let T be a subphylogeny for (Q, α) such that* penalty$(T) \leq q$ *and B be the bad tree of T. Then, $|\bigcup_{v \in B} Im(v)| \leq 4q$.*

Proof. An edge subfamily P at v is imperfect if either (a) P shares a state σ with $S - P$ on character c and $c(v) \neq \sigma$ or (b) $P \cup \{v\}$ does not have a perfect

NEAR-PERFECT-PHYLOGENY(S, C, q).

1. Let $T =$ PERFECT-PHYLOGENY(S, C). If $T \neq$ NIL, then return T.
2. If $|S| \leq qr + 1$, then use exhaustive enumeration to search for a minimum-length q-near-perfect phylogeny for (S, C). Return NIL if no such phylogeny exists. Otherwise, return any such phylogeny.
3. For each $C_p \subseteq C$ such that $|C_p| \geq m - q$, find a minimum-length C_p-perfect phylogeny T_{C_p} of penalty at most q for S, if one exists, as follows:
   (a) Allocate a table $\mathcal{N}$ with one entry for each possible pair (Q, α), where α is a penalty state assignment and Q is a subfamily. Initialize $\mathcal{N}(Q, \alpha)$ to NIL for every pair (Q, α).
   (b) Enumerate, by increasing cardinality of Q, the pairs (Q, α) such that α is a penalty state assignment and Q is a subfamily. For each (Q, α), attempt (using Lemma 9) to find a minimum-length C_p-perfect subphylogeny of penalty at most q. If such a subphylogeny T_Q exists, set $\mathcal{N}(Q, \alpha) = T_Q$.
   (c) Let T_{C_p} be the minimum-length tree from among those that can be obtained by putting an edge between the roots of subphylogenies for (Q_1, α_1) and (Q_2, α_2) such that $Q_2 = S - Q_1$ and $\mathcal{N}(Q_1, \alpha_1), \mathcal{N}(Q_2, \alpha_2) \neq$ NIL.
4. Return the tree T_{C_p} that minimizes length(T_{C_p}) over all sets C_p enumerated in the previous step. If no tree exists, return NIL.

FIG. 3. *The near-perfect phylogeny algorithm.*

phylogeny. The number of subfamilies of the latter sort is at most q, since each of them contributes at least 1 to the total penalty. Let K be the set of subfamilies P that satisfy (a). Let K_0 be the subset of K consisting of all subfamilies P such that there is a character c and species $s \in P$ satisfying the requirements that P is a subfamily at $v \in V(B)$, $c(v) \neq c(s)$, and $c(s) = c(s')$ for some $s' \in S - Q$. Since each $P \in K_0$ contributes at least 1 to the total penalty, $|K_0| \leq q$. Let J be the graph whose vertex set is $K - K_0$ and whose edge set is defined as follows. Let $Q_u \in K - K_0$ and $Q_v \in K - K_0$ be imperfect subfamilies at u and v, respectively. There is an edge between Q_u and Q_v in J if and only if there are a character c and species $s_u \in Q_u$ and $s_v \in Q_v$ such that $c(u) \neq c(s_u) = c(s_v) \neq c(v)$. Let μ be the size of a maximum matching in J. One can verify that

$$q \geq \mu + |K - K_0| - 2\mu \geq |K - K_0|/2.$$

Therefore, $|K - K_0| \leq 2q$, and the lemma follows. □

5. The algorithm. Our near-perfect phylogeny algorithm is shown in Figure 3. Its analysis relies on the result below, proved in the next subsection.

LEMMA 9. *A minimum-length subphylogeny for a pair (Q, α) can be found in $|Q|m^{O(q)}2^{O(q^2r^2)}$ time and $O(q(r + \log m))$ space.*

We now have the main result of this paper.

THEOREM 10. *The algorithm* NEAR-PERFECT-PHYLOGENY *runs in time $|S|m^{O(q)}2^{O(q^2r^2)}$. That is, for fixed q and r, the problem of determining whether S has a q-near-perfect phylogeny, and, if so, finding such a tree of minimum length, can be solved in polynomial time.*

Proof. Step 1 takes $O(2^{2r}nm^2)$ time, as explained in section 3. By Theorem 1, if

S has a perfect phylogeny, this tree must also be an optimum near-perfect phylogeny. It can be shown that step 2 can be completed within the claimed time bound.

We now argue that step 3 of NEAR-PERFECT-PHYLOGENY finds an optimal phylogeny for each choice of C_p. Assume that step 3(b) correctly computes a minimum-length subphylogeny for each pair (Q, α) it considers (or determines that no such tree exists). Let T be any minimum-length C_p-perfect phylogeny for S. It suffices to prove that in step 3(c) the algorithm encounters a C_p-perfect phylogeny T' for S such that $\text{length}(T') = \text{length}(T)$.

By Lemma 4 and the fact that $|S| > qr + 1$, T must have at least one good edge $e = (u_1, u_2)$. For $i = 1, 2$, let α_i be the penalty state assignment of u_i, and let $Q_i = S \cap V(T_{u_i})$. Then, for $i = 1, 2$, Q_i is a character subfamily and, by Lemma 5, (Q_i, α_i) has a subphylogeny T_i' of length at most $\text{length}(T_i)$. Without loss of generality, assume that this T_i' is generated in step 3(b). Then, step 3(c) generates a tree T' by putting an edge between the roots of T_1 and T_2. By the minimality of T, $\text{length}(T') = \text{length}(T)$, as claimed.

Note that NEAR-PERFECT-PHYLOGENY enumerates only sets C_p of size at least $m - q$, because a q-near-perfect phylogeny has at most q nonconvex characters. Thus, the result returned by step 4 is a minimum-length q-near-perfect phylogeny for S, if one exists.

The total number of sets C_p considered in step 3 is

$$\sum_{i=m-q}^{m} \binom{m}{i} = O(qm^q),$$

and step 3(b) enumerates $O(m2^r r^q)$ (Q, α) pairs. By Lemma 9, this leads to a total running time of $|S| m^{O(q)} 2^{O(q^2 r^2)}$. □

5.1. Computing a subphylogeny. We now prove Lemma 9 by giving an algorithm to find a minimum-length C_p-perfect subphylogeny T for (Q, α). The key idea is given by Lemma 6, which suggests that, to find a C_p-perfect subphylogeny T of minimum penalty, it suffices to guess $B = B(T)$ and, for each node v of B, the perfect and imperfect edge subfamilies at v. Our procedure enumerates a sequence of *candidates*, each of which is used to generate a potential tree. A candidate consists of four pieces of information:

- $\tilde{B}$, a guess as to the bad tree of T.
- For each node v of $\tilde{B}$, a penalty state assignment α_v such that $\alpha_v = \alpha$ if v is the root of $\tilde{B}$.
- $\tilde{\mathcal{P}}$, a mapping from each vertex v of $\tilde{B}$ to a subset of S representing a guess as to the union of perfect edge subfamilies at v.
- $\tilde{Im}$, a mapping from each vertex v of $\tilde{B}$ to a collection of subsets of S representing a guess as to the collection of imperfect edge subfamilies at v.

Assume that the candidate is a correct guess as to the various components of a subphylogeny for (Q, α). We now describe how to construct such a subphylogeny from this information.

We first find, for each $v \in \tilde{B}$, the decomposition $\tilde{Pe}(v)$ of $\tilde{\mathcal{P}}(v)$ into perfect edge subfamilies. As in algorithm SUBPHYLOGENY (Figure 2), we rely on Lemma 2, which states that if we know one of the subfamilies R such that $R \subseteq \tilde{\mathcal{P}}(v)$, we have one of two possibilities:

(i) $\tilde{Pe}(v) = \{R, \tilde{\mathcal{P}}(v) - R\}$ or
(ii) $\tilde{Pe}(v) = R \cup (\tilde{\mathcal{P}}(v) - R)/v$, where $c(v) = c(Sv(\tilde{\mathcal{P}}(v), R))$ for every character $c \in C$.

In the latter case, $c(Sv(\tilde{\mathcal{P}}(v), R)) \neq *$ for every character c. There are polynomially many (for fixed r) choices for R; one of these must enable us to make the appropriate decomposition of $\mathcal{P}(v)$ if the candidate is a correct guess.

The distribution of perfect and imperfect edge subfamilies across the vertices of $\tilde{B}$ forces the states of some its nodes to assume certain values in order to maintain the convexity of the corresponding characters. For a vertex v in $\tilde{B}$, let Q_v be the set of all species in the subtree of T rooted at v. The state of vertex v in $\tilde{B}$ on character $c \in C_p$ is forced to equal σ if either Q_v and $S - Q_v$ share a state on character c or there are distinct subtrees T_1, T_2 at v, where T_1, T_2 contain species x_1, x_2, respectively, such that $c(x_1) = c(x_2)$. The remaining unforced states are set in any way that is consistent with convexity of the characters in C_p. This can be done in time polynomial in n, m, and r.

We now produce a subphylogeny for (Q, α) by doing the following for each $v \in \tilde{B}$:

(a) For each $R \in \tilde{Im}(v)$, enumerate all penalty state assignments γ to find the pair (R, γ) such that $T_R = \mathcal{N}(R, \gamma) \neq \emptyset$ and $\text{length}(T_R) + \text{dist}(u, v)$ is minimum, where u is the root of T_R. Connect the root of T_R to v.
(b) For every $R \in \tilde{Pe}(v)$, enumerate all penalty state assignments γ to find a pair (R, γ) such that $T_R = \mathcal{N}(R, \gamma) \neq \emptyset$ and the tree obtained by linking v to the root of T_R is a perfect phylogeny for $R \cup \{v\}$. Connect the root of T_R to v.

There is, of course, no guarantee that a candidate is a correct guess from which a minimum-length subphylogeny can be constructed. Indeed, it is possible that a candidate simply cannot be used to produce a subphylogeny for (Q, α). For instance, a candidate is invalid if some $s \in Q$ is neither a vertex in $\tilde{B}$ nor contained in some set in either $\tilde{Im}(v)$ or $\tilde{Pe}(v)$ for some $v \in V(\tilde{B})$. A candidate is also invalid if it is impossible to make a state assignment for $\tilde{B}$ on the characters in C_p in any consistent way. Finally, in either of steps (a) or (b) above, it may be impossible to find the required subtrees of a node $v \in \tilde{B}$. In any case, if an invalid candidate is encountered, we dismiss it. If no valid candidate can be generated for a (Q, α) pair, we set $\mathcal{N}(Q, \alpha) = \emptyset$.

It is also possible that a candidate allows us to generate a subphylogeny but not a minimum-length one. This issue is resolved by enumerating *all* potential candidates. The tree T stored in $\mathcal{N}(Q, \alpha)$ is the one that minimizes the length among all trees generated from valid candidates.

5.2. Generating candidates. We now describe how candidates are generated and derive a bound on their number. First, observe that, by Lemma 4, we need to consider only trees $\tilde{B}$ with at most qr edges. Thus, $qr^{O(qr)}$ distinct tree topologies $\tilde{B}$ are generated. Enumerating them takes time $qr^{O(qr)}$ and space $O(qr \log qr)$. The number of penalty state assignments enumerated for the nodes in $\tilde{B}$ is $r^{O(q^2 r)}$. These can be generated in time $2^{O(q^2 r^2)}$ and space $O(q^2 r^2)$.

Suppose that $\tilde{B}$ is indeed the bad tree of a subphylogeny T for (Q, α_Q). By Lemma 8, there is some set of at most $4q$ character subfamilies containing all imperfect edge subfamilies at v; each of these is a potential choice for $\tilde{Im}(v)$. There are $m^{O(q)} 2^{O(qr)}$ choices of subfamilies and $(qr)^{O(q)}$ ways in which these subfamilies can be distributed among the vertices of $\tilde{B}$.

By Lemma 7, for every $v \in \tilde{B}$ we can restrict our attention to $\tilde{\mathcal{P}}(v)$ of the form

$\tilde{\mathcal{P}}(v) = G(v) - U$, where

$$U = (S - Q) \cup \{u \in P \colon P \in \tilde{Im}(v), v \in \tilde{B}\} \qquad \text{and} \qquad G(v) = \bigcap_{i=1}^{k} Q_i$$

such that $\{Q_i\}_{i=1}^{k}$ is a set of character subfamilies for different characters $c \in C - C_p$. For every node v in $\tilde{B}$, $G(v)$ is the intersection induced by $\tilde{Im}(v)$.

Thus, the total number of candidates is $m^{O(q)}2^{O(q^2r^2)}$. These can be generated in time $m^{O(q)}2^{O(q^2r^2)}$ and space $O(q(\log m + r))$.

It can be verified that processing a candidate takes time $O(|Q|r^q)$. The total time to find a minimum-penalty subphylogeny for (Q, α) is therefore $|Q|m^{O(q)}2^{O(q^2r^2)}$, and the space used is $O(q(r + \log m))$. This concludes the proof of Lemma 9.

6. Conclusions and open questions. We have shown that a relaxed version of the perfect phylogeny problem, parameterized by the degree q to which the resulting phylogeny deviates from perfection, can be solved in polynomial time. Since the perfect phylogeny model is too restrictive, our algorithm may have some practical use. Unfortunately, its practicality is limited by its running time, which is bounded by a polynomial whose degree depends on q. We note, however, that the time bound is based on the perhaps overly pessimistic assumption that all bad edges can occur together in a bad tree. One may ask whether this is likely to happen in practice. Also, is there a parameter that is smaller than q in practice, in terms of which to express the time bound? One candidate is the maximum size of a bad tree.

Perhaps the most important open question raised by our algorithm is whether it is possible to make the degree of the polynomial describing the running time independent of the parameters; that is, whether there is an algorithm with running time $O(f(q)p(|S|, m))$, where p is a polynomial whose degree does not depend on q. Alternatively, one could try to show that the case where r is fixed and q is the only parameter is hard for $W[1]$.

Acknowledgments. We thank the referee and editor for their suggestions, which substantially improved the presentation.

REFERENCES

[1] R. Agarwala and D. Fernández-Baca, *A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed*, SIAM J. Comput., 23 (1994), pp. 1216–1224.

[2] H. Bodlaender, M. Fellows, and T. Warnow, *Two strikes against perfect phylogeny*, in Proceedings of the 19th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci., Springer-Verlag, 1992, pp. 273–283.

[3] W. H. E. Day, D. S. Johnson, and D. Sankoff, *The computational complexity of inferring rooted phylogenies by parsimony*, Math. Biosci., 81 (1986), pp. 33–42.

[4] G. F. Estabrook, *Cladistic methodology: A discussion of the theoretical basis for the induction of evolutionary history*, Annu. Rev. Ecology and Systematics, 3 (1972), pp. 427–456.

[5] D. Fernández-Baca and J. Lagergren, *A polynomial-time algorithm for near-perfect phylogeny*, in Proceedings of the 23rd International Conference on Automata, Languages, and Programming, Lecture Notes in Comput. Sci., Springer-Verlag, 1996, pp. 670–680.

[6] W. M. Fitch, *Aspects of molecular evolution*, Annu. Rev. Genet., 7 (1973), pp. 343–380.

[7] L. R. Foulds and R. L. Graham, *The Steiner problem in phylogeny is NP-complete*, Adv. Appl. Math., 3 (1982), pp. 43–49.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[9] D. Gusfield, *The Steiner Tree Problem in Phylogeny*, Technical report 334, Computer Science Department, Yale University, New Haven, CT, 1984.
[10] J. Håstad, *Clique is hard to approximate within* $n^{1-\epsilon}$, Acta Math., 182 (1999), pp. 105–142.
[11] S. Hougardy and H. J. Prömel, *A* 1.598 *approximation algorithm for the Steiner problem in graphs*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, SIAM, Philadelphia, 1999, pp. 448–453.
[12] S. Kannan and T. Warnow, *A fast algorithm for the computation and enumeration of perfect phylogenies*, SIAM J. Comput., 26 (1997), pp. 1749–1763.
[13] F. R. McMorris, T. J. Warnow, and T. Wimer, *Triangulating vertex-colored graphs*, SIAM J. Discrete Math., 7 (1994), pp. 296–306.
[14] M. A. Steel, *The complexity of reconstructing trees from qualitative characters and subtrees*, J. Classification, 9 (1992), pp. 91–116.
[15] T. Warnow, D. Ringe, and A. Taylor, *Reconstructing the evolutionary history of natural languages*, in Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, Atlanta, GA, SIAM, Philadelphia, 1996, pp. 314–322.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1128–1139

THE 3-COLORABILITY PROBLEM ON GRAPHS WITH MAXIMUM DEGREE FOUR*

MARTIN KOCHOL†, VADIM LOZIN‡, AND BERT RANDERATH§

Abstract. The 3-colorability problem is known to be NP-complete in the class of graphs with maximum degree four. On the other hand, due to the celebrated theorem of Brooks, the problem has a polynomial-time solution for graphs with maximum degree three. To make the complexity gap more precise, we study a family of intermediate graph classes between these two extremes and classify all of them according to the computational complexity of the problem. In particular, we generalize Brooks's theorem in the case of 3-colorability to a larger class by showing that every connected graph in that class is 3-colorable, unless it is a complete graph on four vertices.

Key words. 3-colorability, polynomial algorithms, NP-completeness, dichotomy

AMS subject classifications. 05C15, 68Q20

DOI. 10.1137/S0097539702418759

1. Introduction.

A 3-*coloring* of a graph is a mapping from the set of vertices to $\{1, 2, 3\}$ such that any two adjacent vertices have different colors. By Brooks's theorem [1] every connected graph with the maximum vertex degree at most three has a 3-coloring or is isomorphic to a complete graph on four vertices K_4. Hence, the decision problem, whether a given graph G has a 3-coloring, is trivial for graphs with maximum degree three. On the other hand, the complexity of the problem jumps from triviality to NP-completeness when we turn to the class of graphs with maximum degree four (see [4]). To make the complexity gap more precise, we study a family of graph classes that are intermediate between these two extremes. Every class $\mathcal{X}$ in the family is associated with a subset $\mathcal{H}'$ of the set of graphs on four vertices and is defined as follows: a graph G belongs to $\mathcal{X}$ if and only if each vertex of G has degree at most four, and the neighborhood of each 4-degree vertex induces a graph isomorphic to a member of the set $\mathcal{H}'$. The main result of the paper is that dichotomy holds: for a given subset $\mathcal{H}'$, the problem either

(1) is NP-complete or

(2) can be solved in linear time.

Subject to the assumption $P \neq NP$ both cases exclude each other. For the linear-time cases we present an algorithm which not only decides existence but also finds a 3-coloring, if there is one. One of the classes in the family is of particular interest because of the fact that every connected graph in that class is 3-colorable, except for a K_4. Thereby, we extend Brooks's theorem in the case of 3-colorability.

Our work is related to [6, 7, 9, 10, 11, 13], where the complexity of finding a 3-coloring in graphs defined by "forbidden induced subgraph" conditions is studied.

*Received by the editors December 3, 2002; accepted for publication (in revised form) May 21, 2003; published electronically August 6, 2003. The results of this paper were obtained while the first author visited RUTCOR and Georgia Tech and the third author visited RUTCOR.

http://www.siam.org/journals/sicomp/32-5/41875.html

†MÚ SAV, Štefánikova 49, 814 73 Bratislava 1, Slovakia (kochol@savba.sk).

‡RUTCOR, Rutgers University, 640 Bartholomew Rd, Piscataway, NJ 08854-8003 (lozin@rutcor.rutgers.edu).

§Institut für Informatik, Universität zu Köln, Pohligstr. 1, 50969 Köln, Germany (randerath@informatik.uni-koeln.de).

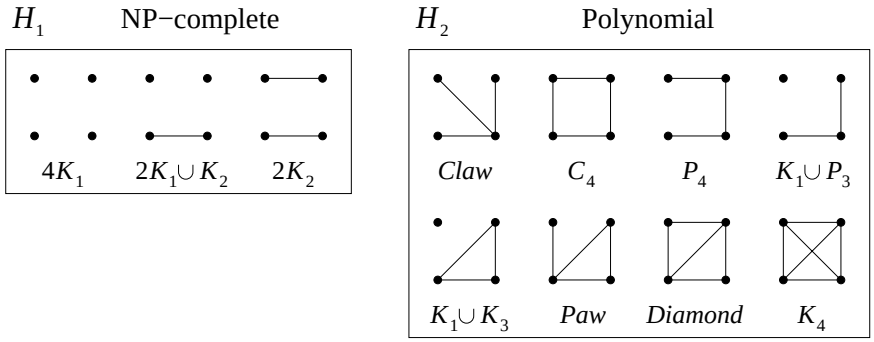


FIG. 2.1. *Problem-specific partition of* $\mathcal{H}$.

2. Basic notations and main result. For graph theoretic terminology not defined in this paper, the reader is referred to [3]. All graphs considered are finite, undirected, and without loops and multiple edges. The set of vertices and the set of edges of a graph G are denoted by $V(G)$ and $E(G)$, respectively. We let $n(G)$ denote the number of vertices of G and $n_4(G)$ denote the number of vertices of degree four in G. For a vertex $v \in V(G)$, we write $N_G(v)$ for the neighborhood of v, i.e., the set of vertices adjacent to v. The closed neighborhood of v is defined as $N_G[v] := N_G(v) \cup \{v\}$. The degree of v is denoted by $d_G(v)$ and the maximum degree of a vertex in G by $\Delta(G)$. If no confusion arises, we omit the subscript G in the above notations.

Given a subset $X \subseteq V(G)$, we denote by $G[X]$ the subgraph of G induced by the vertices in X, and let $G - X := G[V(G) - X]$. If $X = \{v\}$, we write $G - v$ for simplicity. We also write $G - e$ to denote the subgraph of G obtained by deleting an edge $e \in E(G)$. If G contains no induced subgraph isomorphic to a graph H, we say that G is H-free.

Throughout the paper, P_k and C_k stand, respectively, for a chordless path and a chordless cycle on k vertices, and K_r stands for a complete graph on r vertices. In particular, K_3 is a *triangle*. The set of all 4-vertex graphs will be denoted $\mathcal{H}$. This set contains 11 pairwise nonisomorphic graphs represented in Figure 2.1 along with their notations. By $\mathcal{H}_1$, $\mathcal{H}_2$, we refer to the subsets of $\mathcal{H}$ depicted in Figure 2.1.

A 4-vertex graph of particular interest is a *diamond* (see Figure 2.1). The nonadjacent vertices of a diamond will be called its *pick* vertices and the other two vertices the *diagonal* vertices. Importance of this graph for the 3-colorability problem is due to the following simple observation.

PROPOSITION 2.1. *Let G be a 3-colorable graph containing a diamond as an induced subgraph. Then the pick vertices of the diamond have the same color in any 3-coloring of G.*

In this paper we study the following problem. Let $\mathcal{H}'$ be a fixed subset of $\mathcal{H}$.

$\mathcal{H}'$-ISOM-3-COL

Instance: A graph G with maximum degree four in which the neighborhood of each 4-degree vertex induces a graph isomorphic to a member of $\mathcal{H}'$.

Question: Does there exist a 3-coloring of G?

Obviously, if $\mathcal{H}' = \emptyset$, we deal with the 3-colorability problem restricted to graphs with maximum degree three, which is polynomially solvable according to Brooks's theorem. The main result of this paper asserts that $\mathcal{H}'$-ISOM-3-COL is NP-complete *if and only if* $\mathcal{H}'$ contains one of the graphs in the set $\mathcal{H}_1$.

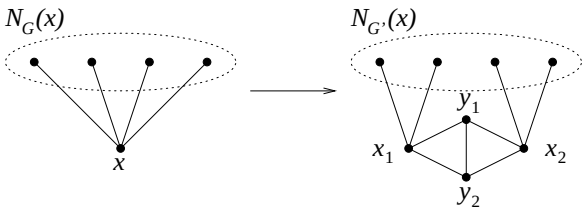


FIG. 2.2. *Reduction for the case* $2K_1 \cup K_2 \in \mathcal{H}'$.

THEOREM 2.2. *The problem* $\mathcal{H}'$-ISOM-3-COL *is*

(i) NP-*complete if* $\mathcal{H}' \cap \mathcal{H}_1 \neq \emptyset$,

(ii) *solvable in linear time if* $\mathcal{H}' \cap \mathcal{H}_1 = \emptyset$.

Proof. Now we prove only (i). The rest is proved in the next two sections.

Obviously, the problem is in NP for any choice of $\mathcal{H}'$. If $4K_1 \in \mathcal{H}'$, then the statement holds by the result of Maffray and Preissmann [9], who proved that the 3-colorability problem is NP-complete for K_3-free graphs with maximum degree four.

Applying the transformation shown in Figure 2.2 to every vertex of degree four in a K_3-free graph G with $\Delta(G) \leq 4$, we obtain a graph G', which is obviously an instance of $\mathcal{H}'$-ISOM-3-COL in the case that $2K_1 \cup K_2 \in \mathcal{H}'$. According to Proposition 2.1, G' is 3-colorable if and only if G is 3-colorable, which implies the statement in this case.

If $2K_2 \in \mathcal{H}'$, we refer the reader to the result of Holyer [5], who has shown that it is an NP-complete problem to decide whether a cubic K_3-free graph G is 3-*edge*-colorable. This result can be translated to the terminology of vertex colorability by associating with each graph G its line graph, denoted $L(G)$ and defined as follows: the vertex set of $L(G)$ is $E(G)$, and two vertices are adjacent in $L(G)$ if and only if the respective edges of G have a vertex in common. It is not hard to verify that if G is a triangle-free cubic graph, then $L(G)$ is a 4-regular diamond- and claw-free graph. Therefore, the result of Holyer implies NP-completeness of vertex 3-colorability in the class of 4-regular diamond- and claw-free graphs. Moreover, we may suppose that any graph G in this class is K_4-free, since a K_4 is not 3-colorable. But now G is an instance of the problem in question, and the conclusion holds. □

3. Polynomial-time results. Let $\mathcal{R}$ denote the class of graphs every vertex of which has degree at most four, and the neighborhood of each 4-degree vertex induces a graph isomorphic to a member of $\mathcal{H}_2$. If $G \in \mathcal{R}$, then $A(G)$ denotes the set of 4-degree vertices x in G such that $G[N(x)] \neq K_1 \cup P_3$.

LEMMA 3.1. *Suppose we have a graph* $G \in \mathcal{R}$ *together with* $A(G)$. *If* $A(G) \neq \emptyset$, *then we can construct in a constant time a graph* $G^* \in \mathcal{R}$ *together with* $A(G^*)$ *such that* $n(G^*) < n(G)$, G^* *is* 3-*colorable if and only if* G^* *is* 3-*colorable, and every* 3-*coloring* φ^* *of* G^* *can be transformed into a* 3-*coloring* φ *of* G *in a constant time.*

Proof. In order to construct the desired graph G^* we use finitely many operations of vertex/edge deletion/addition or set $G^* = K_4$. A trivial consequence of this is that $A(G)$ can be transformed into $A(G^*)$ in a constant time as well.

Let $x \in A(G)$. If $G[N(x)]$ contains a triangle, G is not 3-colorable, and we can set $G^* = K_4$. If $G[N(x)]$ contains no triangle, then exactly one of the following cases may occur.

Case 1. $G[N(x)]$ is a claw. Let $N(x) = \{y_1, y_2, y_3, x'\}$ and x' have degree three in $G[N(x)]$. Denote by G^* the graph obtained from G by deleting the vertices of $N[x]$ and introducing a new vertex y adjacent to those vertices in $V(G) - N[x]$ that have

a neighbor in $\{y_1, y_2, y_3\}$.

Clearly, $n(G^*) < n(G)$. Assume φ is a 3-coloring of G. Then, according to Proposition 2.1, y_1, y_2, y_3 have the same color, and we may transform φ into a 3-coloring of G^* by assigning to the vertex y the value $\varphi(y_1)$. The inverse direction follows by analogy. Thus G^* is 3-colorable if and only if G^* is, and every 3-coloring φ^* of G^* can be transformed into a 3-coloring φ of G in a constant time.

To prove that $G^* \in \mathcal{R}$, let us notice that each y_j has at most one neighbor in $V(G) - N[x]$, since otherwise $N(y_j)$ would induce in G a member of $\mathcal{H}_1$. This implies that y has degree at most three in G^*. Obviously, the degree of any other vertex of G^* is at most four. Suppose now that G^* contains a vertex z of degree four such that $N_{G^*}(z)$ induces a graph $H \in \mathcal{H}_1$. Notice that $G[N_G(z)]$ either coincides with H if $N_G(z) \cap \{y_1, y_2, y_3\} = \emptyset$ or is isomorphic to a spanning subgraph of H otherwise. In both cases, $G[N_G(z)] \in \mathcal{H}_1$, contradicting the assumption.

Case 2. $G[N(x)] = C_4$. Let $N(x) = \{y_0, y_1, y_2, y_3\}$ and y_i be adjacent to y_{i+1} for $i = 0, 1, 2, 3 \pmod 4$. Denote by G^* the graph obtained from G by deleting the vertices of $N[x]$, introducing a pair of adjacent vertices y_{odd} and y_{even}, and connecting y_{odd} (respectively, y_{even}) to those vertices in $V(G) - N[x]$ that have a neighbor in the set $\{y_1, y_3\}$ (respectively, $\{y_0, y_2\}$) in G. The proof of the lemma in this case is similar to the proof of Case 1.

Case 3. $G[N(x)] = P_4$. Let $N(x) = \{y_0, y_1, y_2, y_3\}$ and y_i be adjacent to y_{i+1} for $i = 0, 1, 2$. Denote by G' the graph obtained from G by deleting the vertices of $N[x]$, introducing a pair of adjacent vertices y_{odd} and y_{even}, and connecting y_{odd} (respectively, y_{even}) to those vertices in $V(G) - N[x]$ that have a neighbor in the set $\{y_1, y_3\}$ (respectively, $\{y_0, y_2\}$) in G. Clearly G' has a 3-coloring if and only if G does, and $n(G') < n(G)$. Furthermore, it is not difficult to verify that

(*) $\Delta(G') \leq 4$ and for every vertex $z \in V(G) - N[x]$ with $d_{G'}(z) = 4$, the neighborhood $N_{G'}(z)$ induces a member of $\mathcal{H}_2$.

If $G' \in \mathcal{R}$, then $G^* = G'$ has the desired properties.

Let $G' \notin \mathcal{R}$. Then by (*), the neighborhood of y_{odd} or y_{even} induces in G' a graph isomorphic to a member of $\mathcal{H}_1$. Assume

(0) y_{even} has degree four in G' with $N(y_{even}) = \{y_{odd}, z_0, z_1, z_2\}$ for some vertices $z_0, z_1, z_2 \in V(G) - N[x]$. Without loss of generality let $N_G(y_0) = \{x, y_1, z_0, z_1\}$ and $N_G(y_2) = \{x, y_1, y_3, z_2\}$.

(Note that the case when y_{odd} has degree four in G' is similar.) If the subgraph of G' induced by $N(y_{even})$ belongs to $\mathcal{H}_1$, then we have the following:

(1) z_1 (or z_0) is adjacent to y_1 in G, and hence to y_{odd} in G', since otherwise the neighborhood of y_0 would induce in G a graph isomorphic to a member of $\mathcal{H}_1$; thus $N_G(y_1) = \{x, y_0, y_2, z_1\}$.

(2) y_3 is adjacent neither to z_0 nor to z_2, since otherwise the subgraph of G' induced by $N(y_{even})$ does not belong to $\mathcal{H}_1$; for the same reason, z_1 is adjacent neither to z_0 nor to z_2.

(3) The subgraph of G' induced by $N(y_{odd})$ does not belong to $\mathcal{H}_1$, since otherwise z_2 should be adjacent to y_3 by analogy with (1).

Under conditions (0)–(3), let us define G^* to be the graph obtained from G by deleting the vertices of the set $U := \{x, y_1, y_3, z_1\}$, introducing a pair of adjacent vertices w and y_{odd}, and connecting y_{odd} (respectively, w) to those vertices in $V(G) - U$ that have a neighbor in the set $\{y_1, y_3\}$ (respectively, $\{x, z_1\}$) in G.

We check that $G^* \in \mathcal{R}$. It is obvious that for any vertex $z \in V(G) - U$, $d_{G^*}(z) \leq d_G(z)$. In order to show that w has degree at most four in G^*, let us observe that

z_1 has degree at most three in G, since otherwise $N_G(z_1)$ would induce in G a graph isomorphic to a member of $\mathcal{H}_1$. Denoting a possible third neighbor of z_1 in G (different from y_0 and y_2) by v, we therefore conclude that w has at most four neighbors in G^*: y_0, y_2, y_{odd}, and v. Similarly, y_3 has degree at most three in G, and hence y_{odd} is of degree at most four in G^*. In the subgraph H of G^* induced by $N(w)$ the vertex y_{odd} has degree at least two, and therefore $H \notin \mathcal{H}_1$. Analogously, the subgraph of G^* induced by $N(y_{odd})$ also is not a member of $\mathcal{H}_1$. For any other vertex z of the graph G^*, we use the same arguments as before: if z has degree four in G^*, then z has degree four in G as well. Moreover, the subgraph $G[N_G(z)]$ is isomorphic to a spanning subgraph of the graph induced by the neighborhood of z in G^*. Therefore, $G^* \notin \mathcal{R}$ would imply $G \notin \mathcal{R}$. To check the other desired properties for G^* is straightforward. □

4. Dart graphs. By means of reductions from Lemma 3.1, any graph $G \in \mathcal{R}$ can be transformed in linear time into a graph G' with $\Delta(G') \leq 4$ such that the neighborhood of each 4-degree vertex in G' induces either a $K_1 \cup P_3$ or a graph containing a triangle. In the latter case, G' obviously is not a 3-colorable graph, since it contains a K_4. Therefore, according to Lemma 3.1, the graph G is not 3-colorable too. In the present section we fix the remaining case; i.e., we solve the problem for graphs with maximum degree four in which the neighborhood of each 4-degree vertex induces a graph isomorphic to $K_1 \cup P_3$. In other words, each vertex of degree four together with its neighborhood induces the graph depicted in Figure 4.1. This graph is known in the literature under the name *dart*. Thus, we call the graphs in the class under consideration *dart graphs*.

The class of dart graphs is an extension of the graphs with maximum degree three. According to the classical result of Brooks [1], every connected graph with vertex degree at most three is 3-colorable, unless it is a K_4. It turns out that this proposition remains valid in the larger class of dart graphs.

In a dart graph G, a vertex v will be called a diagonal (respectively, pick) vertex if v is a diagonal (respectively, pick) vertex of an induced diamond in G.

LEMMA 4.1. *In a dart graph G, the subsets of pick and diagonal vertices are disjoint. As a result,*

(a) *in G no pick vertex has degree more than three;*
(b) *in the subgraph of G induced by the diagonal vertices, no vertex has degree more than two.*

Proof. Let u be a pick vertex of an induced diamond D in G with the diagonal (the edge connecting the diagonal vertices of D) v_1v_2. By contradiction, assume uw is the diagonal of another induced diamond D' in G. Notice that w neither coincides with v_i nor is adjacent to v_i, since otherwise $G[N[v_i]]$ is not a dart ($i = 1, 2$). But then u has at least five neighbors in G: v_1, v_2, w, and the two pick vertices of D'. This contradiction shows that the subsets of pick and diagonal vertices are disjoint in G.

By definition, every vertex of degree four in G is a diagonal vertex, which implies (a). Every diagonal vertex has at least two nondiagonal (pick) neighbors, which implies (b). □

Consider a connected component C of the graph induced by the diagonal vertices of a dart graph G. Due to Lemma 4.1(b), C is either a chordless path or a chordless cycle. By adding to C all the diamonds containing the vertices of C we obtain a subgraph of G (not necessarily induced) that will be denoted by F. If F contains a vertex which has degree four in G, we call F a *diamond bracelet*. The number of diamonds in the bracelet will be called the *size* of the bracelet. An example of a

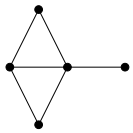


FIG. 4.1. *The dart.*

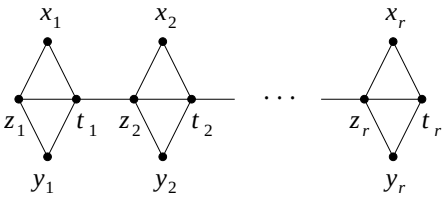


FIG. 4.2. *A diamond bracelet.*

bracelet of size r is depicted in Figure 4.2. Notice that the vertices z_1 and t_r may have degree three in G. They also may be adjacent if C is a cycle. We denote the family of diamond bracelets of G by $B(G)$. Notice that $B(G) = \emptyset$ if and only if $n_4(G) = 0$.

LEMMA 4.2. *Suppose we have a K_4-free dart graph G together with $B(G)$. If $n_4(G) > 0$, then we can construct in a constant time a K_4-free dart graph G^* together with $B(G^*)$ such that $n(G^*) \le n(G)$, $n_4(G^*) < n_4(G)$, and every 3-coloring φ^* of G^* can be transformed into a 3-coloring φ of G in a constant time.*

Proof. In order to construct the desired graph G^* we use finitely many operations of vertex/edge deletion/addition. A trivial consequence of this is that $B(G)$ can be transformed into $B(G^*)$ in a constant time as well. In most cases it is straightforward to see that G^* is a K_4-free dart graph. When this fact is not that obvious, we provide necessary explanations.

The lemma is trivial if G contains a vertex of degree four with a neighbor v of degree at most two: in this case $G^* = G - v$, and φ^* can be extended to φ by assigning to v a color missing in the neighborhood of v. Thus, in the course of the proof we shall assume that every neighbor of a 4-degree vertex has degree at least three.

Since $n_4(G) > 0$, we may consider a diamond bracelet $F \in B(G)$ of size $r \ge 1$. We use the notation of vertices of F as indicated in Figure 4.2. In our case analysis we distinguish the following seven cases that exhaust all possibilities for F.

1. $r \ge 3$.
2. $r \in \{1, 2\}$, and at least one of the vertices z_1 and t_r has degree three.

If $r \in \{1, 2\}$ and both vertices z_1 and t_r have degree four, we denote by t_0 and z_{r+1} the neighbors of z_1 and t_r different from x_1, t_1, y_1 and x_r, z_r, y_r, respectively. With these notations, we divide the general case when z_1 and t_r have degree four into the following subcases:

3. $r = 2$ and $z_1 t_2 \in E(G)$ (i.e., $t_0 = t_2$ and $z_3 = z_1$).
4. $r = 2$, $z_1 t_2 \notin E(G)$, and $t_0 \ne z_3$.
5. $r = 2$ and $t_0 = z_3$.
6. $r = 1$, and t_0, z_2 have no common neighbor of degree four.
7. $r = 1$, and t_0, z_2 have a common neighbor of degree four.

Moreover, cases 5 and 7 are subdivided into further subcases depending on certain specific conditions arising in the proof.

Having outlined the case analysis, let us proceed with the proof according to our plan.

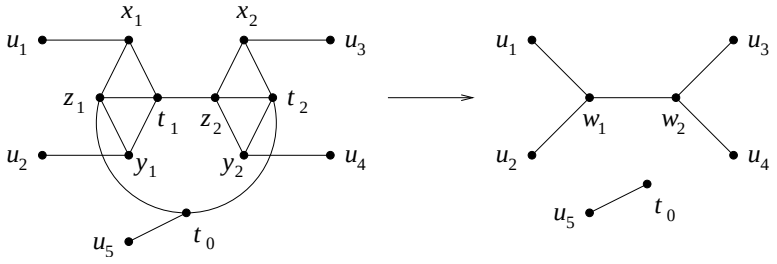


FIG. 4.3.

Case 1. $r \geq 3$. Let G^* be the K_4-free dart graph arising from G by deleting the edges t_1z_2, t_2z_3 and adding a new edge t_1z_3. Then $n(G^*) = n(G)$, $n_4(G^*) = n_4(G) - 2$ and for any 3-coloring φ^* of G^* the colors $\varphi^*(z_2)$ and $\varphi^*(t_2)$ are different and can interchange. Moreover, the colors $\varphi^*(t_1)$ and $\varphi^*(z_3)$ also are different. Therefore, by possible switching of colors of the vertices z_2 and t_2 we can transform φ^* into φ so that $\varphi(t_1) \neq \varphi(z_2)$ and $\varphi(t_2) \neq \varphi(z_3)$, which means φ is a 3-coloring of G.

Case 2. $r \in \{1, 2\}$, and at least one of the vertices z_1 and t_r has degree three. Assume t_r is of degree three. Then z_r has degree four in G (otherwise F does not cover any vertex of degree four). Let u denote the fourth neighbor of z_r, i.e., $u \neq t_r, x_r, y_r$. The vertices z_r and u are not the neighbors of a same 4-degree vertex. Therefore, $G^* = G - z_ru$ is a K_4-free dart graph, and furthermore $n(G^*) = n(G)$, $n_4(G^*) \leq n_4(G) - 1$. If φ^* is a 3-coloring of G^*, then, clearly, the colors $\varphi^*(z_r)$ and $\varphi^*(t_r)$ are different and can interchange, and hence we may transform φ^* into a 3-coloring φ of G in a constant time.

From now on, let $N(z_1) = \{t_0, x_1, y_1, t_1\}$ and $N(t_r) = \{z_r, x_r, y_r, z_{r+1}\}$.

Case 3. $r = 2$ and $z_1t_2 \in E(G)$ (i.e., $t_0 = t_2$ and $z_1 = z_3$). Consider $G^* = G - \{z_1t_2, z_2t_1\}$, which is clearly a K_4-free dart graph with $n(G^*) = n(G)$, $n_4(G^*) = n_4(G) - 4$. Since the colors $\varphi^*(z_1)$ and $\varphi^*(t_1)$ (and similarly $\varphi^*(z_2)$ and $\varphi^*(t_2)$) are different and can interchange, we conclude that φ^* can be transformed into φ in a constant time.

Case 4. $r = 2$, $z_1t_2 \notin E(G)$, and $t_0 \neq z_3$. If both t_0 and z_3 belong to F, say, $t_0 = x_2$ and $z_3 = x_1$, consider the graph $G^* = G - \{z_1, t_1, x_1, z_2, t_2, x_2\}$. Then $n(G^*) = n(G) - 6$, $n_4(G^*) = n_4(G) - 4$, and φ^* can be easily extended to a 3-coloring of G in a constant time.

If $z_3 \notin F$, then we apply exactly the same arguments as in Case 1: transform G into G^* by deleting the edges t_1z_2, t_2z_3 and adding a new edge t_1z_3, and then transform φ^* into φ by color interchanging, analogously if $t_0 \notin F$.

Case 5. $r = 2$ and $t_0 = z_3$. Remember that each of the vertices x_1, y_1, x_2, y_2, t_0 has degree exactly three. Let us denote their respective neighbors different from z_1, t_1, z_2, t_2 by u_1, u_2, u_3, u_4, u_5 (see Figure 4.3). Notice that $t_0 \neq u_1, \ldots, u_4$, but some of the vertices $u_1, \ldots, u_5$ may coincide.

Transform G into a graph G' by deleting the vertices of F and introducing a pair of new vertices w_1, w_2 with $N(w_1) = \{u_1, u_2, w_2\}$ and $N(w_2) = \{u_3, u_4, w_1\}$ (see Figure 4.3).

Case 5.1. Assume first that G' is a K_4-free dart graph. Then set $G^* = G'$. Now $n(G^*) = n(G) - 6$, $n_4(G^*) = n_4(G) - 4$, and φ^* can be transformed into φ in the following way. If at least two values among $\varphi^*(u_1)$, $\varphi^*(u_2)$, $\varphi^*(u_5)$ are equal, we can assign to the vertices x_1, y_1, t_0 the same color which is not in the set $\{\varphi^*(u_1), \varphi^*(u_2), \varphi^*(u_5)\}$.

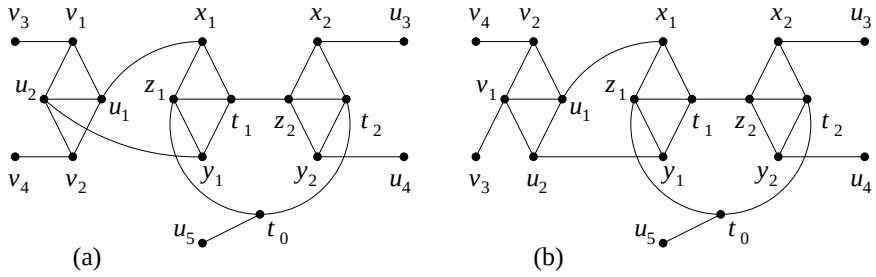


FIG. 4.4.

The rest is trivial: define $\varphi(x_2) = \varphi(y_2) \notin \{\varphi^*(u_3), \varphi^*(u_4)\}$, $\varphi(t_2) \notin \{\varphi(x_2), \varphi(t_0)\}$, $\varphi(z_2) \notin \{\varphi(x_2), \varphi(t_2)\}$, $\varphi(t_1) \notin \{\varphi(z_2), \varphi(x_1)\}$, $\varphi(z_1) \notin \{\varphi(x_1), \varphi(t_1)\}$. If $\varphi^*(u_1)$, $\varphi^*(u_2)$, $\varphi^*(u_5)$ are pairwise different, then at least two values among $\varphi^*(u_3)$, $\varphi^*(u_4)$, $\varphi^*(u_5)$ must be equal (otherwise φ^* is not a 3-coloring of G^*), and we can apply symmetrical arguments.

Case 5.2. Let G' be a dart graph containing a K_4. Then both new vertices w_1 and w_2 belong to the K_4 (indeed, $w_1 \notin K_4$ implies $w_2 \notin K_4$, since w_2 has at most two neighbors except w_1, but then G contains the same K_4). Therefore, up to symmetry, $u_1 = u_3$, $u_2 = u_4$, and $u_1u_2 \in E(G)$. Under this assumption, u_1 and u_2 cannot have degree four in G, and hence t_0 is a cut-point of G. Setting $G^* = G' - \{w_1, w_2, u_1, u_2\}$ we obtain a K_4-free dart graph G^* such that $n(G^*) = n(G) - 10$, $n_4(G^*) = n_4(G) - 4$. A 3-coloring φ^* of G^* can be extended to a 3-coloring φ of G as follows: $\varphi(z_1) = \varphi(z_2) = \varphi(u_1) \neq \varphi^*(t_0)$, $\varphi(t_1) = \varphi(t_2) = \varphi(u_2) \notin \{\varphi^*(t_0), \varphi(z_1)\}$, $\varphi(x_i) = \varphi(y_i) = \varphi^*(t_0)$, $i = 1, 2$.

Case 5.3. Suppose that G' is not a dart graph. Then at least one of the vertices $u_1, \ldots, u_4$, say u_1, has degree four in G', but $G'[N[u_1]]$ is not a dart. Therefore,

- u_1 has degree four in G;
- $u_1 \neq u_3, u_4$; otherwise $G[N[u_1]]$ is not a dart; as a result, $u_1w_2 \notin E(G')$;
- $u_1 \neq u_2$, and u_1 is adjacent to u_2; otherwise $G[N[u_1]]$ is not a dart;
- u_1 is not adjacent to t_0; otherwise $G[N[u_1]]$ is not a dart.

Denote the other two neighbors of u_1 in G, different from u_2 and x_1, by v_1 and v_2. A straightforward analysis reveals two possible combinations of the vertices in the neighborhood of u_1 in the graph G, shown in Figure 4.4. Notice that vertices v_3 and v_4 must exist in both cases. In the case of Figure 4.4(a), this follows from our assumption that every vertex in the neighborhood of u_1 has degree at least three. In the case of Figure 4.4(b), the vertex v_3 exists, since otherwise we could apply the arguments of Case 2 to the diamond induced by u_1, u_2, v_1, v_2.

Assume first that $v_3 = t_0$ (or, equivalently, $v_1 = u_5$). Then the subgraph H of G induced by the vertices of F together with u_1, u_2, v_1, v_2, t_0 has at most three neighbors in the remaining part of G: u_3, u_4, and v_4. In this case, we define $G^* = G - V(H)$ and extend φ^* to φ according to the following rules: $\varphi(x_2) = \varphi(y_2) = \varphi(t_0) \notin \{\varphi^*(u_3), \varphi^*(u_4)\}$, $\varphi(x_1) = \varphi(y_1) = \varphi(v_2) \notin \{\varphi^*(v_4), \varphi(x_2)\}$. The rest is simple. The case $v_4 = t_0$ can be analyzed by analogy.

If neither v_3 nor v_4 coincides with t_0, then we define G^* to be the graph obtained from G by deleting the vertices u_1, u_2, v_1, v_2 and adding new edges x_1v_3 and y_1v_4. Let us show that G^* is a dart graph without a K_4. The K_4-freeness is obvious. It is also easy to see that every vertex of G^* has degree at most four. Assume now that G^* contains a vertex w of degree four such that $G^*[N[w]]$ is not a dart. Then

$w \notin \{v_3, v_4\}$; otherwise $G[N[w]]$ also is not a dart. Therefore, the dart $G[N[w]]$ has been destroyed under the transformation $G \to G^*$ because of a new edge connecting two vertices adjacent to w, say y_1 and v_4. However, the vertices y_1 and v_4 have no common neighbor of degree four in G, unless $v_4 = t_0$. Thus G^* is the desired graph with $n(G^*) = n(G) - 4$, $n_4(G^*) = n_4(G) - 2$. A 3-coloring φ^* of G^* can be extended to a 3-coloring φ of G in the following way. For the configuration in Figure 4.4(a), define $\varphi(v_1) = \varphi(v_2) = \varphi^*(x_1)$, $\varphi(u_1) \neq \varphi^*(x_1)$, and $\varphi(u_2) \notin \{\varphi(u_1), \varphi^*(x_1)\}$. For the configuration in Figure 4.4(b), define $\varphi(u_1) = \varphi^*(v_4)$, $\varphi(v_1) = \varphi^*(x_1)$, $\varphi(u_2) = \varphi(v_2) \notin \{\varphi^*(x_1), \varphi^*(v_4)\}$.

Case 6. $r = 1$, and t_0, z_2 have no common neighbor of degree four. If t_0, z_2 have two common neighbors v_1, v_2 of degree three which are adjacent to each other, then $G^* = G - \{t_0, z_2, v_1, v_2\}$ is a K_4-free dart graph with $n(G^*) = n(G) - 4$, $n_4(G^*) = n_4(G) - 2$, and we can trivially extend φ^* to φ in a constant time. Suppose that t_0, z_2 have no common adjacent neighbors of degree three. Then deleting the edges t_0z_1, t_1z_2 and adding a new edge t_0z_2 (if $t_0z_2 \notin E(G)$) results in a K_4-free dart graph G^* such that $n(G^*) = n(G)$, $n_4(G^*) = n_4(G) - 2$. In any 3-coloring φ^* of G^* the colors of z_1, t_1 are different and can interchange, and the colors of t_0, z_2 are different. Therefore, φ^* can be transformed into a 3-coloring φ of G so that $\varphi(t_1) \neq \varphi(z_2)$ and $\varphi(z_1) \neq \varphi(t_0)$.

Case 7. $r = 1$, and t_0, z_2 have a common neighbor of degree four. Then, there must exist another diamond bracelet F' containing that neighbor. Without loss of generality we may suppose that the size of F' is 1, since otherwise we can apply to F' the arguments in Cases 1–5. Moreover, as for F, we assume that both diagonal vertices of F' have degree four, and we denote the vertices of F' and the respective neighbors of its diagonal vertices by analogy with F, i.e., $z'_1, t'_1, x'_1, y'_1, t'_0, z'_2$. Up to symmetry, we may suppose that the common neighbor of t_0 and z_2 is z'_1, and z_2 coincides with x'_1. For t_0, there remain two possibilities: either $t_0 = y'_1$, in which case we say that F *strongly depends* on F', or $t_0 = t'_0$, in which case we say that F *weakly depends* on F'. To avoid Case 6 for F', we conclude that there is a bracelet F'' of size 1 (else apply Cases 1–5 to F'') such that F' depends on F'' either strongly or weakly.

Case 7.1. F depends on F' weakly. Observe that in this case F' must depend on F'' weakly. Indeed, if F' depends on F'' strongly, then, according to the definition, t'_0 is a pick vertex of F'', and the neighborhood of t'_0 contains z_1, z'_1, and the two diagonal vertices of F'', contradicting Lemma 4.1.

Case 7.1.1. $z'_2 \in F$. Then $F = F''$ (see Figure 4.5(a)). In this case we obtain G^* by deleting from G all vertices of F and F' together with t_0. Clearly, G^* is a K_4-free dart graph with $n(G^*) = n(G) - 9$, $n_4(G^*) \leq n_4(G) - 4$. Given a 3-coloring φ^* of G^*, we can always extend it to a 3-coloring φ of G in such a way that $\varphi(y_1) = \varphi(y'_1) \notin \{\varphi^*(u_1), \varphi^*(u_2)\}$ and $\varphi(t_0) \notin \{\varphi^*(u_3), \varphi(y_1)\}$. The rest is trivial.

Case 7.1.2. $z'_2 \notin F$. With the help of Lemma 4.1 we conclude that F'' is disjoint from $F \cup F'$. Moreover, we may suppose that

- no diagonal vertex of F'' has a neighbor in F'; otherwise we are in conditions of Case 7.1.1 for the pair of bracelets F' and F'';
- one of the diagonal vertices of F'' is adjacent to t_0 by the definition of weak dependency;
- the other diagonal vertex of F'' is adjacent to a pick vertex of F; otherwise Case 6 can be applied to F''.

Therefore, we have the situation indicated in Figure 4.5(b). We define G^* to be

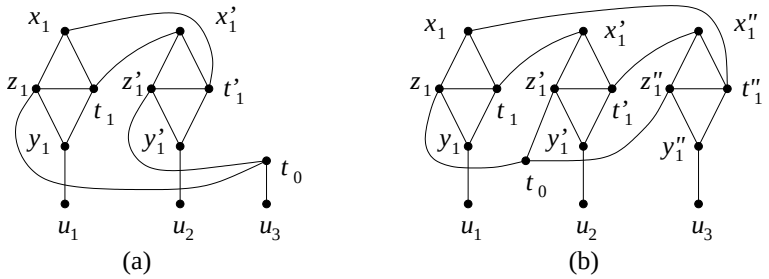


FIG. 4.5.

the graph obtain from G by deleting all vertices of F, F', and F" along with t_0. Given a 3-coloring φ^* of G^*, we can always extend it to a 3-coloring φ of G in such a way that $\varphi(y_1) = \varphi(y_1') \notin \{\varphi^*(u_1), \varphi^*(u_2)\}$, $\varphi(y_1'') \notin \{\varphi(y_1), \varphi^*(u_3)\}$, and $\varphi(t_0) \notin \{\varphi(y_1), \varphi(y_1'')\}$. The rest is trivial.

Case 7.2. F depends on F' strongly. If F' depends on F'' weakly, we are in conditions of Case 7.1 for the pair of bracelets F', F''. If F' depends on F'' strongly and $F \neq F''$, we delete from G the vertices of F' and connect the diagonal vertices of F to the pick vertices of F'', obtaining the desired K_4-free dart graph G^* with $n(G^*) = n(G) - 4$, $n_4(G^*) = n_4(G) - 2$. Any 3-coloring of G^* can be extended to a 3-coloring of G by coloring the vertices of F' analogously to the vertices of F. If $F = F''$, then $G[V(F \cup F')]$ is a 3-colorable component of G, and hence the graph $G^* = G - V(F \cup F')$ has the required properties. □

THEOREM 4.3. *A dart graph G is 3-colorable if and only if it has no component isomorphic to a K_4. If G is 3-colorable, then a 3-coloring of G can be found in linear time.*

Proof. The necessity of the first part of the theorem is trivial. To see the sufficiency, observe that a dart graph is K_4-free if and only if it has no component isomorphic to a K_4. The same is true if G is a graph with vertex degree at most three. Therefore, the sufficiency follows from Lemma 4.2 and Brooks's theorem [1].

We can check whether a dart graph G is K_4-free in linear time. Analogously, the set of 4-degree vertices and the set $B(G)$ of diamond bracelets of G can be found in linear time. Consequently, by means of Lemma 4.2 we can create in linear time a K_4-free graph G' without vertices of degree more than three such that any 3-coloring of G' can be transformed into a 3-coloring of G in linear time. By [8] (see also [12]), a 3-coloring of G' can be found in linear time, which proves the statement. □

End of the proof of Theorem 2.2. In linear time we can decide whether $G \in \mathcal{R}$ and construct $A(G)$. By Lemma 3.1, in linear time we can construct a graph G' such that $n(G') \leq n(G)$, $A(G') = \emptyset$, and G' is 3-colorable if and only if G is 3-colorable. If G' contains a K_4, which can be checked in linear time for graphs of bounded degree, then both G' and G are not 3-colorable. Otherwise, G' is a K_4-free dart graph, and hence both G' and G have a 3-coloring by Theorem 4.3. □

5. Outline of the algorithm. Let us summarize the above arguments in the following linear-time procedure, 3COL4MAX4, that decides not only existence but also finds a 3-coloring, if there is one, for any graph $G \in \mathcal{R}$.

PROCEDURE 3COL4MAX4.

Input: Graph $G \in \mathcal{R}$.

Output: "NO" if G is not 3-colorable; a 3-coloring of G otherwise.

(1) **While** G has a vertex v such that $G[N(v)]$ is either a claw or a C_4 or a P_4, apply reductions from Lemma 3.1.
(2) **If** G contains a K_4 as an induced subgraph, then output "NO" and STOP.
(3) Apply reductions of Lemma 4.2 to produce a graph of vertex degree at most three, color this graph, and then transform the found 3-coloring into a 3-coloring of the input graph as suggested by Lemmas 4.2 and 3.1.

The correctness and linear-time bound of the procedure follow from the results of the preceding sections, where we described in detail all steps of the algorithm, except for 3-coloring of graphs with vertex degree at most three. The algorithmic proof of Brooks's theorem proposed by Lovász [8] suggests an idea how to find a 3-coloring in such graphs. Recent developments in [12] provide a simpler way to implement Lovász's algorithm in linear time. To make the paper self-contained we present the algorithmic proof of Brooks's theorem below. Bryant [2] simplified this proof with the following characterization of cycles and complete graphs, highlighting thereby the exceptional role of the cycles and complete graphs in Brooks's theorem.

PROPOSITION 5.1. *Let G be a 2-connected graph. Then G is a cycle or a complete graph if and only if $G - \{u, v\}$ is not connected for every pair $\{u, v\}$ of vertices at distance two.*

Proof (Randerath and Schiermeyer [10]). Let G be a 2-connected graph of order n. If G is a cycle or a complete graph, then obviously $G - \{u, v\}$ is not connected for any pair $\{u, v\}$ of vertices at distance two. Hence, assume that G is neither a cycle nor a complete graph and that $G - \{u, v\}$ is not connected for each pair $\{u, v\}$ of vertices at distance two. Since G is 2-connected, there must exist at least one cycle in G. Let C be a longest cycle in G. Assume that C is not a Hamiltonian cycle of G. Since C is a longest cycle and G is connected, there exist vertices y, z of C and $x \in V(G) - V(C)$ such that z is adjacent to x and y, and x is not adjacent to y; i.e., $\mathrm{dist}_G(x, y) = 2$. Denote the path connecting x to y and containing all the vertices of C by P_1. According to our assumption, $G - \{x, y\}$ is not connected, and due to the 2-connectivity of G, there exists a second path P_2 connecting x to y, which is vertex disjoint with P_1, except for the endpoints. But then P_1 and P_2 form a cycle C' of length greater than C, a contradiction to the special choice of C. Thus, $C = v_0v_1, \ldots, v_{n-1}v_0$ is a Hamiltonian cycle. Now we consider a vertex v_i with $2 < d_G(v_i) < n - 1$, which must exist since G is neither a cycle nor a complete graph. Then there is $j \notin \{i - 1, i + 1\}$ such that v_i is adjacent to v_j but nonadjacent (without loss of generality) to v_{j-1}. Since $G - \{v_i, v_{j-1}\}$ is not connected, v_j is not adjacent to v_{j-2}. Therefore, $G - \{v_j, v_{j-2}\}$ is not connected, and hence $d_G(v_{j-1}) = 2$. Specifically, v_{j-1} is not adjacent to v_{j+1}. But now, on the one hand, $G - \{v_{j-1}, v_{j+1}\}$ is not connected according to our assumption, and, on the other hand, v_j is adjacent to v_i. This contradiction completes the proof of the proposition. □

THEOREM 5.2 (Brooks [1]). *Let G be a connected graph which is neither an odd cycle nor a complete graph. Then G is $\Delta(G)$-colorable.*

Algorithmic proof of Brooks's theorem [8]. Obviously, if the graph G with $\Delta(G) = \Delta$ is a cycle of even length, then G is 2-colorable. Hence, assume that G is neither a complete graph nor a cycle. Moreover, we shall assume that G is 2-connected, since otherwise the problem can be reduced to blocks of G. By Proposition 5.1 there exists a pair $\{u, v\}$ of vertices at distance two in graph G such that $G' = G - \{u, v\}$ is connected. Suppose that $w \in N_G(u) \cap N_G(v)$, and label the vertices of G as $v_1 = u$, $v_2 = v$, and the remainder as $v_3, v_4, \ldots, v_n = w$ in nonincreasing order of their distance from w in G'. Then color the vertices $v_1, v_2, \ldots, v_n$ in that order using

colors $1, 2, \ldots, \Delta$ so that at each step the color used for the vertex v_i is the lowest-numbered color not yet used at a vertex adjacent to v_i. Hence, we apply a greedy algorithm along our vertex ordering. The labeling of the vertices ensures that at each stage, v_i $(1 \leq i \leq n)$ is adjacent to at least one higher numbered (and hence presently uncolored) vertex. Thus, v_i is adjacent to at most $\Delta - 1$ colored vertices, and one of the Δ colors will be available for it. Finally, the vertex $v_n = w$ is adjacent to at most Δ vertices, at least two of which ($v_1 = u$ and $v_2 = v$) have the same color 1. Hence, there will be a color available for w. Therefore, the algorithm provides a vertex-coloring of G with at most Δ colors. □

REFERENCES

[1] R. L. Brooks, *On coloring the nodes of a network*, Proc. Cambridge Phil. Soc., 37 (1941), pp. 194–197.

[2] V. Bryant, *A characterization of some* 2*-connected graphs and a comment on an algorithmic proof of Brooks' theorem*, Discrete Math., 158 (1996), pp. 279–281.

[3] R. Diestel, *Graph Theory*, Springer-Verlag, New York, 2000.

[4] M. R. Garey, D. S. Johnson, and L. Stockmeyer, *Some simplified* NP*-complete graph problems*, Theoret. Comput. Sci., 1 (1976), pp. 237–267.

[5] I. Holyer, *The* NP*-completeness of edge-coloring*, SIAM J. Comput., 10 (1981), pp. 718–720.

[6] M. Kochol, *Linear algorithm for* 3*-coloring of locally connected graphs*, in Experimental and Efficient Algorithms, Lecture Notes in Comput. Sci. 2647, K. Jansen, M. Margraf, M. Mastrolilli, and J. D. P. Rolim, eds., Springer-Verlag, New York, 2003, pp. 191–194.

[7] D. Král, J. Kratochvíl, Zs. Tuza, and G. Woeginger, *Complexity of coloring without forbidden induced subgraphs*, in Graph-Theoretic Concepts in Computer Science, Lecture Notes in Comput. Sci. 2204, A. Brandstädt and V. B. Le, eds., Springer-Verlag, New York, 2001, pp. 263–271.

[8] L. Lovász, *Three short proofs in graph theory*, J. Combin. Theory Ser. B, 19 (1975), pp. 269–271.

[9] F. Maffray and M. Preissmann, *On the* NP*-completeness of the k-colorability problem for triangle-free graphs*, Discrete Math., 162 (1996), pp. 313–317.

[10] B. Randerath and I. Schiermeyer, *A note on Brooks' theorem for triangle-free graphs*, Australas. J. Combin., 26 (2002), pp. 3–10.

[11] B. Randerath, I. Schiermeyer, and M. Tewes, *Three-colourability and forbidden subgraphs.* II: *Polynomial algorithms*, Discrete Math., 251 (2002), pp. 137–153.

[12] S. Skulrattanakulchai, Δ*-list vertex coloring in linear time*, in Algorithm Theory—SWAT 2002, Lecture Notes in Comput. Sci. 2368, M. Penttonen and E. Meineche Schmidt, eds., Springer-Verlag, New York, 2003, pp. 240–248.

[13] I. Zverovich, *Coloring of Locally-Connected Graphs*, RUTCOR Research Report 32-2002, Rutgers University, Piscataway, NJ, 2002, http://rutcor.rutgers.edu/∼rrr.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1140–1157

ALGORITHMS FOR BOOLEAN FUNCTION QUERY PROPERTIES*

SCOTT AARONSON†

Abstract. We investigate efficient algorithms for computing Boolean function properties relevant to query complexity. Such properties include, for example, deterministic, randomized, and quantum query complexities; block sensitivity; certificate complexity; and degree as a real polynomial. The algorithms compute the properties, given an n-variable function's truth table (of size $N = 2^n$) as input.

Our main results are the following:

- $O(N^{\log_2 3} \log N)$ algorithms for many common properties,
- an $O(N^{\log_2 5} \log N)$ algorithm for block sensitivity,
- an $O(N)$ algorithm for testing "quasi symmetry,"
- a notion of a "tree decomposition" of a Boolean function, proof that the decomposition is unique, and an $O(N^{\log_2 3} \log N)$ algorithm for finding it,
- a subexponential-time approximation algorithm for space-bounded quantum query complexity. To develop this algorithm, we give a new way to search systematically through unitary matrices using finite-precision arithmetic.

The algorithms discussed have been implemented in a linkable library.

Key words. algorithm, Boolean function, truth table, query complexity, quantum computation

AMS subject classifications. 68Q10, 68Q17, 68Q25, 68W01, 81P68

DOI. 10.1137/S0097539700379644

1. Introduction.

The query complexity of Boolean functions, also called black-box or decision-tree complexity, has been well studied for years [7, 9, 18]. Counting how many queries are needed to evaluate a function is easier than counting how many computational steps are needed; thus, nontrivial lower bounds are more readily shown for the former measure than for the latter. Also, query complexity has proved to be a powerful tool for studying the capabilities of quantum computers [2, 3, 4, 7, 13].

Numerous Boolean function properties relevant to query complexity have been defined, such as sensitivity, block sensitivity, randomized and quantum query complexity, and degree as a real polynomial. But many open questions remain concerning the relationships between the properties. For example, are sensitivity and block sensitivity polynomially related? How small can quantum query complexity be, relative to randomized query complexity? Lacking answers to these questions, we may wish to gain insight into them by using computer analysis of small Boolean functions. But to perform such analysis, we need efficient algorithms to compute the properties in question. Such algorithms are the subject of the present paper.

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, and let $N = 2^n$ be the size of the truth table of f. We seek algorithms that have modest running time as a function of N, given the truth table as input. Table 1 lists some properties important for query complexity, together with the complexities of the most efficient algorithms for them of which we know.

*Received by the editors October 16, 2000; accepted for publication (in revised form) April 17, 2003; published electronically August 6, 2003. This work was supported by a California MICRO Fellowship, an NSF Graduate Fellowship, and the Defense Advanced Research Projects Agency (DARPA) and Air Force Laboratory, Air Force Materiel Command, USAF, under agreement F30602-01-2-0524.

http://www.siam.org/journals/sicomp/32-5/37964.html

†Computer Science Division, UC Berkeley, Berkeley, CA 94720-1776 (aaronson@cs.berkeley.edu). This work was done at Bell Laboratories, Murray Hill, NJ.

TABLE 1

Properties important for query complexity, with the complexities of the most efficient current algorithms. Here "LP" stands for linear programming reduction, and "SDP" for semidefinite programming reduction.

Query property	**Complexity**	**Source**
Deterministic query complexity $\mathrm{D}(f)$	$O(N^{1.585} \log N)$	[12]
Certificate complexity $\mathrm{C}(f)$	$O(N^{1.585} \log N)$	[10]
Degree as a real polynomial $\deg(f)$	$O(N^{1.585} \log N)$	This paper
Approximate degree $\widetilde{\deg}(f)$	LP	Obvious
Randomized query complexity $\mathrm{R}_0(f)$	LP	This paper
Block sensitivity $\mathrm{bs}(f)$	$O(N^{2.322} \log N)$	This paper
Quasi symmetry	$O(N)$	This paper
Tree decomposition	$O(N^{1.585} \log N)$	This paper
Quantum query complexity $\mathrm{Q}_2(f)$	SDP	[5]
Randomized certificate complexity $\mathrm{RC}(f)$	LP	Obvious

There is also a complexity-theoretic rationale for studying algorithmic problems such as those considered in this paper. Much effort has been devoted to finding Boolean function properties that do not naturalize in the sense of Razborov and Rudich [20], and that might therefore be useful for proving circuit lower bounds. In our view, it would help this effort to have a better general understanding of the complexity of problems on Boolean function truth tables—both upper and lower bounds. This paper is a step towards such an understanding.

In an earlier version of this paper, we raised as our "most interesting" open question whether there exists a polynomial-time algorithm to compute (or even approximate) quantum query complexity. Subsequently Barnum, Saks, and Szegedy [5] managed to answer this question in the affirmative, using semidefinite programming. In section 7 we present a weaker result, namely an $O(N^{\mathrm{polylog}(N)})$ constant-factor approximation algorithm for bounded-error quantum query complexity if the memory of the quantum computer is restricted to $O(\log n)$ qubits. The main issue here, which is only partly addressed by results of Bernstein and Vazirani [8], is how to represent unitary operators with limited-precision arithmetic. Our results on this issue are used in the detailed analysis of the algorithm of Barnum, Saks, and Szegedy.

We have implemented most of the algorithms discussed in this paper in a linkable C library, which is available for download.[1]

The paper is organized as follows. Section 2 gives preliminaries, and section 3 reviews simple algorithms for deterministic query complexity, certificate complexity, degree as a real polynomial, randomized query complexity, and randomized certificate complexity. Section 4 gives an $O(N^{\log_2 5} \log N)$ algorithm for computing block sensitivity, and section 5 gives an $O(N)$ algorithm for testing "quasi symmetry." Section 6 defines a notion of the "tree decomposition" of a Boolean function, proves that the decomposition is unique, and gives an $O(N^{\log_2 3} \log N)$ algorithm for constructing it. Section 7 presents our results on algorithms for quantum query complexity, and section 8 concludes with some open problems.

2. Preliminaries. A Boolean function f is a total function from $\{0,1\}^n$ onto $\{0,1\}$. We use V_f to denote the set of variables of f, and use X, or alternatively $x_1, \ldots, x_n$, to denote an input to f. The restriction of f to R is denoted $f_{|R}$. If X is an input, $|X|$ denotes the Hamming weight of X; if S is a set, $|S|$ denotes the

[1] Available at http://www.cs.berkeley.edu/~aaronson/bfw

cardinality of S. Particular Boolean functions to which we refer are AND_n, OR_n, and XOR_n, the AND, OR, and XOR functions, respectively, on n inputs.

Throughout, we assume a RAM model of computation, in which for any input X, $f(X)$ can be obtained in $O(1)$ time.

3. Basic properties. To our knowledge, no algorithms for block sensitivity, quasi symmetry, or tree decomposition have been previously published. But algorithms for simpler query properties have appeared in the literature.

3.1. Deterministic query complexity. A *decision tree* T is a binary tree in which each nonleaf vertex is labeled with an index (1 through n) and each leaf vertex is labeled with an output (0 or 1). Evaluation begins at the root. At a vertex v labeled with i, if $x_i = 0$, we evaluate the left subtree of v, while if $x_i = 1$, we evaluate the right subtree. When we reach a leaf vertex we halt and return the appropriate output. T represents a Boolean function f if, for all settings of $x_1, \ldots, x_n$, the output of T equals $f(x_1, \ldots, x_n)$. Then the deterministic query complexity $\mathrm{D}(f)$ is the minimum height of a decision tree for f.

Guijarro, Lavín, and Raghavan [12] give a simple $O(N^{1.585} \log N)$ dynamic programming algorithm to compute $\mathrm{D}(f)$. We present a similar algorithm for completeness. The idea is that, at any time, a decision tree for f has reduced f to one of its 3^n possible restrictions: each of the n variables either (1) has been queried and is a 0, (2) has been queried and is a 1, or (3) has not yet been queried. Thus we can represent a restriction S by an element of $\{0, 1, *\}^n$, where the asterisk represents "not yet queried." We can also impose a lexicographic ordering on restrictions by stipulating that 0 comes before 1 comes before asterisk.

The algorithm consists of two loops, both of which proceed through all states in lexicographic order. The first loop fills in an array called A, which stores, for each restriction, whether it is a constant function and, if so, what its value is. The second loop uses A to fill in an array called D, which stores the deterministic query complexity of each restriction. In the algorithm, $S(i)$ represents the ith element of S, and $S_{S(i)=k}$ represents S with $S(i)$ set to the value k.

ALGORITHM 1 (computes deterministic query complexity).

loop over all $S \in \{0, 1, *\}^n$ *in lexicographic order* {
  if $(S \in \{0, 1\}^n)$, *then set* $A[S] := f(S)$ *else* {
    choose an i *such that* $S(i) = *$;
    if $(A[S_{S(i)=0}] = A[S_{S(i)=1}])$, *then set* $A[S] := A[S_{S(i)=0}]$;
    else set $A[S] :=$ NONCONSTANT;
  }
}
loop over all $S \in \{0, 1, *\}^n$ *in lexicographic order* {
  if $(A[S] \neq$ NONCONSTANT$)$, *then set* $D[S] := 0$;
  else set $D[S] := 1 + \min_{S(i)=*}[\max(D[S_{S(i)=0}], D[S_{S(i)=1}])]$;
}
return $D[*^n]$;

That f is given as a truth table is crucial: if f is nontotal and only the inputs for which f is defined are given, then deciding whether $\mathrm{D}(f) \leq k$ for some k is NP-complete [16]; a related problem is NP-hard even to approximate within a constant factor [23].

3.2. Certificate complexity. Given an input X to f, a *certificate* for X is a constant-valued restriction that agrees with X on the fixed variables. The *size* of

the certificate is the number of fixed variables; note that querying these variables is sufficient to prove that $f(X) = 0$ or $f(X) = 1$, as the case may be. The certificate complexity $\mathrm{C}_X(f)$ of X is the minimum size of any certificate for X. The certificate complexity $\mathrm{C}(f)$ of f is the maximum of $\mathrm{C}_X(f)$ over all inputs X. (Equivalently, $\mathrm{C}(f)$ is the minimum height of a nondeterministic decision tree for f.)

Czort [10] gives an $O(N^{1.585} \log N)$ algorithm to compute $\mathrm{C}(f)$. We can obtain such an algorithm by reusing the same array A that was used for deterministic query complexity in section 3.1. Consider a directed acyclic graph G in which the vertices are the 3^n possible restrictions of f, and an edge is drawn from S to T if and only if T is obtained from S by changing one 0 or 1 to an asterisk. The main loop of the algorithm fills in an array called q, proceeding in *reverse* lexicographic order. The array q stores, for each restriction S of f, the maximum length of any path in G from S to a nonconstant function, given that the path must halt once it reaches a nonconstant function. (Therefore, if S itself is nonconstant, then the length must be 0.) The certificate complexity is obtained by taking the maximum, over all $S \in \{0,1\}^n$, of $n - q[S] + 1$.

ALGORITHM 2 (computes certificate complexity).

loop over all $S \in \{0,1,*\}^n$ *in reverse lexicographic order* {
    if $(A[S] = \text{NONCONSTANT})$, *then*
        set $q[S] := 1 + \max(-1, \max_{S(i) \in \{0,1\}} q[S_{S(i)=*}])$;
    else set $q[S] := 1 + \max_{S(i) \in \{0,1\}} q[S_{S(i)=*}]$;
}
return $\max_{S \in \{0,1\}^n}(n - q[S])$;

Again, if f is not given as a full truth table, then deciding whether $\mathrm{C}(f) \leq k$ for some k is NP-complete [14].

3.3. Degree as a polynomial. Let $\deg(f)$ be the minimum degree of an n-variate real multilinear polynomial p such that, for all $X \in \{0,1\}^n$, $p(X) = f(X)$. Degree was introduced to query complexity by Nisan and Szegedy [19], who observed the relationship $\deg(f) \leq \mathrm{D}(f)$. Later Beals et al. [7] related degree to quantum query complexity by showing that $\deg(f) \leq 2\,\mathrm{Q}_E(f)$.

The following lemma, adapted from Lemma 4 of [9], is easily seen to yield an $O(n3^n) = O(N^{1.585} \log N)$ dynamic programming algorithm for $\deg(f)$. Say that a function obeys the *parity property* if the number of inputs X with odd parity for which $f(X) = 1$ equals the number of inputs X with even parity for which $f(X) = 1$.

LEMMA 3.1 (folklore). $\deg(f)$ *equals the size of the largest restriction of* f *for which the parity property fails.*

Proof. Let c_S be the coefficient of the monomial $\prod_{v_k \in S} v_k$. By the Möbius formula,

$$c_S = \sum_{X \subseteq S} (-1)^{|X|+|S|} f(X),$$

where $X \subseteq S$ means that every i for which $x_i = 1$ is in S. Now f has degree less than d if and only if, for all S with $|S| \geq d$, $c_S = 0$. But for each fixed value of $|S|$, $|S|$ (in the formula above) can affect the sign of c_S but not whether $c_S = 0$. Therefore for all δ, $c_S = 0$ for all S with $|S| = \delta$ is equivalent to the parity property holding for all restrictions of size δ. ∎

Lemma 3.1 leads to an $O(N^{\log_2 3} \log N)$ dynamic programming algorithm for computing $\deg(f)$, as follows.

ALGORITHM 3 (computes degree as a real polynomial).
loop over all $S \in \{0,1,*\}^n$ *in lexicographic order* {
    if $(S \in \{0,1\}^n)$*, then set* $d[S] := f(S)$*;*
    else set $d[S] := d[S_{S(i)=0}] - d[S_{S(i)=1}]$ *for some* i *such that* $S(i) = *$*;*
}
return $\max_{d[S]\neq 0}$(*number of* $*$*'s in* S*);*

3.4. Randomized query complexity. A randomized decision tree T_R is simply a collection $T_1, \ldots, T_k$ of ordinary decision trees, each T_i associated with a probability p_i satisfying $p_1 + \cdots + p_k = 1$. T_R represents f if each tree T_i in the collection represents f. Let $h(T, X)$ be the number of queries tree T makes on input X. Then we define

$$h(T_R) = \max_{X \in \{0,1\}^n} [p_1 h(T_1, X) + \cdots + p_k h(T_k, X)].$$

Then the zero-error randomized query complexity $\mathrm{R}_0(f)$ is the minimum height of a randomized decision tree that represents f. One can also discuss the bounded-error randomized query complexity $\mathrm{R}_2(f)$, which is the minimum height of a randomized decision tree that represents f with a probability of error at most 1/3. Nisan showed that $\mathrm{R}_0(f)^2 \geq \mathrm{D}(f)$ and $\mathrm{R}_2(f)^3 = \Theta(\mathrm{D}(f))$ (see [18]). On the other hand, the best known separation between deterministic and randomized query complexity is $\mathrm{R}_0(f) = \mathrm{R}_2(f) = \mathrm{D}(f)^{0.753\ldots}$ [21, 22] for f an AND/OR tree with two children per node.

Whether better separations are possible is a long-standing open question, and one that might be fruitfully investigated with computer analysis.[2] Unfortunately, though, we do not know how to compute $\mathrm{R}_0(f)$ or $\mathrm{R}_2(f)$ in polynomial time without reliance on linear programming. Here we sketch the reduction to a linear program.

As before, at any time the query algorithm has reduced f to one of its 3^n restrictions. Also, for each restriction, the algorithm has up to $n+2$ possible moves: it can query any variable not yet queried, halt and return 0, or halt and return 1. So consider a directed acyclic graph in which the vertices are the restrictions $S_1, \ldots, S_{3^n}$ (together with halting states $S_{(0)}$ and $S_{(1)}$) and the edges $(S_{i_1}, S_{j_1}), \ldots, (S_{i_m}, S_{j_m})$ are the possible moves of the algorithm. With each edge e we associate a probability weight $p(e)$; these weights are the variables of the linear program. Let $C(X)$ be the subset of N restrictions that are compatible with input X. There are four classes of constraints:

1. *Well-formedness.* The sum of the probability weights leaving the initial state must be 1. Formally $\sum_i p(S_0, S_i) = 1$, where S_0 is the initial state (no variables yet queried).
2. *Conservation of probability.* The sum of the probability weights entering each state must equal the sum of the probability weights leaving it. For all $j \neq 0$, $\sum_i p(S_i, S_j) = \sum_k p(S_j, S_k)$.
3. *Probability of correctness.* For each input, the probability of returning the correct answer must be at least $1 - \epsilon$, where $\epsilon = 0$ for $\mathrm{R}_0(f)$ and $\epsilon = 1/3$ for $\mathrm{R}_2(f)$. For all X with $f(X) = 1$, $\sum_{S_i \in C(X)} p(S_i, S_{(1)}) \geq 1 - \epsilon$. For $f(X) = 0$, substitute $p(S_i, S_{(0)})$.

[2] We have done such analysis for all 4-variable Boolean functions dependent on all 4 inputs. The two functions exhibiting the largest deterministic/randomized complexity gap ($D(f) = 4$, $R_0(f) = 3$) are both AND/OR trees, namely, A AND (B OR C OR D) and (A OR B) AND (C OR D). Randomization yields at least some speedup for 60 out of the 208 Boolean functions that are distinct up to negating inputs and outputs and permuting inputs.

4. *Minimum running time.* For each input, the expected running time must be at most T. For all X, $\sum_{S_i \in C(X)} Q(S_i)[p(S_i, S_{(0)}) + p(S_i, S_{(1)})] \leq T$, where $Q(S_i)$ is the number of queries that have been made in state S_i.

The objective is to minimize T.

Finally, let us briefly mention randomized and quantum certificate complexity. Given $X \in \{0,1\}^n$, Aaronson [1] defined the randomized certificate complexity $\mathrm{RC}_X(f)$ to be the minimum expected number of queries used by a randomized algorithm that accepts with 2/3 probability, given X as input, and rejects with 2/3 probability, given any Y for which $f(Y) \neq f(X)$. Then $\mathrm{RC}(f)$ is the maximum of $\mathrm{RC}_X(f)$ over all X. The quantum certificate complexity $\mathrm{QC}(f)$ is defined analogously, but with quantum instead of randomized algorithms. Aaronson showed that for any X, the optimal (up to a constant factor) randomized certificate for X queries the input nonadaptively. Furthermore, $\mathrm{QC}(f)$ is exactly characterized (again up to a constant factor) as the square root of $\mathrm{RC}(f)$. From these results, it readily follows that both $\mathrm{RC}(f)$ and $\mathrm{QC}(f)$ are approximable to within a constant factor in polynomial time, using linear programming.

4. Block sensitivity. Block sensitivity, introduced by Nisan [18], is a Boolean function property that is used to establish lower bounds. There are several open problems that an efficient algorithm for block sensitivity might help to investigate [18, 7, 9].

Let X be an input to Boolean function f, and let B (a *block*) be a nonempty subset of V_f. Let $X(B)$ be the input obtained from X by flipping the bits of B.

DEFINITION 4.1. *A block B is* sensitive *on X if $f(X) \neq f(X(B))$, and* minimal *on X if B is sensitive and no proper subblock S of B is sensitive. Then the block sensitivity $\mathrm{bs}_X(f)$ of X is the maximum number of disjoint minimal (or equivalently, sensitive) blocks on X. Finally $\mathrm{bs}(f)$ is the maximum of $\mathrm{bs}_X(f)$ over all X.*

The obvious algorithm for computing $\mathrm{bs}(f)$ (compute $\mathrm{bs}_X(f)$ for each X using dynamic programming, then take the maximum) uses $\Theta(N^{2.585} \log N)$ time. Here we show how to reduce the complexity to $O(N^{2.322} \log N)$ by exploiting the structure of minimal blocks. Our algorithm has two main stages: one to identify minimal blocks and store them for fast lookup, another to compute $\mathrm{bs}_X(f)$ for each X using only minimal blocks. The analysis proceeds by showing that no Boolean function has too many minimal blocks, and therefore that if the algorithm is slow for some inputs (because of an abundance of minimal blocks), then it must be faster for other inputs.

ALGORITHM 4 (computes $\mathrm{bs}(f)$). *For each input X do the following:*

1. *Construct an array M of all minimal blocks of X. To do this, loop over all blocks B in lexicographic order ($\{x_1\}, \{x_2\}, \{x_1, x_2\}, \{x_3\}, \ldots$), and mark* (i) *whether B is a minimal block and* (ii) *whether B contains a minimal block. B is minimal if B is sensitive and, for all $x_i \in B$, $B \setminus \{x_i\}$ does not contain a minimal block. B contains a minimal block if B is minimal or $B \setminus \{x_i\}$ contains a minimal block for some $x_i \in B$.*
2. *Create $2^n - 1$ lists, one list L_S for each nonempty subset S of variables. Then, for each minimal block B in M, insert a copy of B into each list L_S such that $B \subseteq S$. The result is that, for each S, $L_S = 2^S \cap M$, where 2^S is the power set of S.*
3. *Let a* state *be a partition (P, Q) of V_f. The set P represents a union of disjoint minimal blocks that have already been selected; the set Q represents the set of variables not yet selected. Then $\mathrm{bs}_X(f) = \theta(\emptyset, V_f)$, where $\theta(P, Q)$ is defined via the recursion $\theta(P, Q) \stackrel{\triangle}{=} 1 + \max_{B \in L_Q} \theta(P \cup B, Q \setminus B)$. Here the*

maximum evaluates to 0 *if* L_Q *is empty. Compute* $\theta(P,Q)$ *using depth-first recursion, caching the values of* $\theta(P,Q)$ *so that each needs to be computed only once.*

The block sensitivity is then the maximum of $\mathrm{bs}_X(f)$ over all X.

Let $m(X,k)$ be the number of minimal blocks of X of size k. The analysis of Algorithm 4's running time depends on the following lemma, which shows that large minimal blocks are rare in any Boolean function.

LEMMA 4.2. $\sum_X m(X,k) \leq 2^{n-k}\binom{n}{k}$ *for* $k \geq 2$.

Proof. The number of positions that can be occupied by a minimal block of size k is $\binom{n}{k}$ for each input, or $2^n\binom{n}{k}$ for all inputs. Consider an input X with a minimal block $B = \{b_1, \ldots, b_k\}$ of size k. Block B has $2^k - 1$ nonempty subsets; label them $S_1, \ldots, S_{2^k-1}$. By the minimality of B, for each S_i the input $X(S_i)$ has $\{b_1\}, \ldots, \{b_k\}$ as minimal blocks if $S_i = B$, and $B \setminus S_i$ as a minimal block if $S_i \neq B$. Therefore, since $k \geq 2$, $X(S_i)$ cannot have B as a minimal block. So of the $2^n\binom{n}{k}$ positions, only one out of 2^k can be occupied by a minimal block of size k. □

THEOREM 4.3. *Algorithm* 4 *takes* $O(N^{2.322}\log N)$ *time.*

Proof. Step 1 takes time $O(N^2 \log N)$, totaled over all inputs. Let us analyze step 2, which creates the $2^n - 1$ lists L_S. Since each minimal block B is contained in $2^{n-|B|}$ sets of variables, the total number of insertions is at most

$$\sum_X \sum_{k=0}^{n} m(X,k)2^{n-k} = \sum_{k=0}^{n}\left[2^{n-k}\sum_X m(X,k)\right] \leq \sum_{k=0}^{n} 2^{2n-2k}\binom{n}{k} = N^{\log_2 5}.$$

Since each insertion takes $O(\log N)$ time, the total time is $O(N^{2.322}\log N)$.

We next analyze step 3, which computes block sensitivity using the minimal blocks. Each $\theta(P,Q)$ evaluation is performed at most once and involves looping through a list of minimal blocks contained in Q, with each iteration taking $O(\log n)$ time. For each block B, the number of distinct (P,Q) pairs such that $B \subseteq Q$ is at most $2^{n-|B|}$. Therefore, by the previous calculation, the time for each input X is at most $(\log N)\sum_{k=0}^{n} m(X,k)2^{n-k}$, and a bound of $O(N^{2.322}\log N)$ follows. □

5. Quasi symmetry. A Boolean function $f(X)$ is *symmetric* if its output depends only on $|X|$. Query complexity is well understood for symmetric functions: for all nonconstant symmetric f, $\mathrm{D}(f) = n$, $\mathrm{R}_0(f) = \Theta(n)$, and $\mathrm{Q}_E(f) = \Theta(n)$ [7]. Thus, a program for analyzing Boolean functions might first check whether a function is symmetric, and if it is, dispense with many expensive tests. We call f *quasi-symmetric* if some subset of input bits can be negated to make f symmetric. There is an obvious $O(N^2)$ algorithm to test quasi symmetry; here we give a linear-time algorithm.

For an integer p, call a restriction of f a *p-left-restriction* if each variable v_i is fixed if and only if $i \leq p$. The basic idea of the algorithm is to loop through all $2^{n+1}-1$ such restrictions, with p decreasing from n to 0. Given a p-left-restriction S, let S_0 and S_1 be the two $(p+1)$-left-restrictions that agree with S. If either $f_{|S_0}$ or $f_{|S_1}$ is not quasi-symmetric, then $f_{|S}$ (and hence f) cannot be quasi-symmetric. If, on the other hand, $f_{|S_0}$ and $f_{|S_1}$ are both quasi-symmetric, then the algorithm tries to fit them together in such a way that $f_{|S}$ itself is seen to be quasi-symmetric. If the fitting-together process succeeds, then the algorithm returns a structure $g[S]$, containing both the output of $f_{|S}$ (encoded in compact form, as a symmetric function) and the *direction*, meaning the set of input bits that must be flipped to make $f_{|S}$ symmetric. Note that $g[S]$ occupies only $O(n-p)$ bits of space. Most of the algorithm deals with the special cases that $f_{|S}$ is an XOR function or a constant function; in both cases $f_{|S}$ is symmetric no matter

which set of input bits is flipped. In the pseudocode, these cases are handled using the tags XOR (for an XOR function), CONSTANT (for a constant function), and NORMAL (for any other quasi-symmetric function). Whenever the algorithm fails (meaning that f has been found not to be quasi-symmetric), the whole algorithm terminates; whenever $g[S]$ is assigned a value, the current iteration terminates. To avoid ambiguity about whether $f_{|S}$ is an XOR, CONSTANT, or NORMAL function, we start p at $n-2$ rather than n.

ALGORITHM 5 (tests quasi symmetry).

For all p-left-restrictions S for $p \le n-2$, with p decreasing from $n-2$ to 0,

1. *If $p = n-2$, then let $g[S]$ be the appropriate* NORMAL, CONSTANT, *or* XOR *function for the 2-input Boolean function $f_{|S}$. If $f_{|S}$ is not quasi-symmetric, then fail (meaning f is not quasi-symmetric).*
2. *If $g[S_0]$ and $g[S_1]$ are* NORMAL *functions but have different directions, then fail.*
3. *Let 0^k be a string of k zeroes. If $g[S_0]$ and $g[S_1]$ are* CONSTANT *functions, then*
   - *If $f_{|S_0}(0^{n-p-1}) = f_{|S_1}(0^{n-p-1})$, then let $g[S]$ be a* CONSTANT *function with output $f_{|S_0}(0^{n-p-1})$; otherwise fail.*
4. *If $g[S_0]$ and $g[S_1]$ are* XOR *functions, then*
   - *If $f_{|S_0}(0^{n-p-1}) \neq f_{|S_1}(0^{n-p-1})$, let $g[S]$ be an* XOR *function with $f_{|S}(0^{n-p}) = f_{|S_0}(0^{n-p-1})$; otherwise fail.*
5. *For $i \in \{0,1\}$, if $g[S_i]$ is a* CONSTANT *function and $g[S_{1-i}]$ is an* XOR *function, then halt and return failure.*
6. *For $i \in \{0,1\}$, if $g[S_i]$ is a* CONSTANT *or* XOR *function, then make $g[S_i]$ a* NORMAL *function with the same direction as $g[S_{1-i}]$.*
7. *For $i \in \{0,1\}$ and $j \in \{0,\dots,n-p-1\}$, let $a_j^{(i)} = f_{|S_i}(X)$ for all $X \in \{0,1\}^{n-p-1}$ of Hamming distance j from the direction string.*
   - *If the strings $a^{(0)}$ and $a^{(1)}$ overlap each other on $n-p-2$ bits, so that for either $i=0$ or $i=1$,*

$$a_1^{(i)} = a_2^{(1-i)}, \dots, a_{n-p-2}^{(i)} = a_{n-p-1}^{(1-i)},$$

   then let $g[S]$ be a NORMAL *function with outputs described by the $(n-p)$-bit overlap string and appropriate direction. Otherwise fail.*

Since the time used by each invocation is linear in $n-p$, the total time used is

$$\sum_{p=0}^{n} 2^p (n-p) = O(N).$$

The following lemma shows that the algorithm deals with all of the ways in which a function can be quasi-symmetric, which is key to the algorithm's correctness.

LEMMA 5.1. *Let f be a Boolean function on n inputs. If two distinct (and noncomplementary) sets of input bits A and B can be flipped to make f symmetric, then f is either* XOR_n, $1-\mathrm{XOR}_n$, *or a constant function.*

Proof. Assume without loss of generality that B is empty. Then A has cardinality less than n. We know that $f(X)$ depends only on $|X|$, and also that it depends only on $/X/ \stackrel{\triangle}{=} \sum_{i=1}^{n} \kappa(x_i)$, where $\kappa(x) = 1-x$ if $x_i \in A$ and $\kappa(x) = x$ otherwise. Choose any Hamming weight $0 \le w \le n-2$, and consider an input Y with $|Y| = w$ and with two variables v_i and v_j such that $v_i \in A$, $v_j \notin A$, and $Y(i) = Y(j) = 0$. Let Z be Y

with $Y(i) = Y(j) = 1$. We have $|Z| = |Y| + 2$, but, on the other hand, $/Z/ = /Y/$, and thus $f(Y) = f(Z)$ by symmetry. Again applying symmetry, $f(P) = f(Q)$ whenever $|P| = w$ and $|Q| = w + 2$. Therefore f is either XOR_n, $1 - \mathrm{XOR}_n$, or a constant function. □

6. Tree decomposition. Many of the Boolean functions of most interest to query complexity are naturally thought of as trees of smaller Boolean functions: for example, AND-OR trees and majority trees. Thus, given a function f, one of the most basic questions we might ask is whether it has a tree decomposition and, if so, what it is. In this section we define a sense in which every Boolean function has a unique tree decomposition, and we prove its uniqueness. We also sketch an $O(N^{1.585} \log N)$ algorithm for finding the decomposition.

DEFINITION 6.1. *A* distinct variable tree *is a tree in which*

(i) *every leaf vertex is labeled with a distinct variable (which may or may not be negated);*
(ii) *every nonleaf vertex v is labeled with a Boolean function having one variable for each child of v, and depending on all of its variables;*
(iii) *every nonleaf vertex has at least two children.*

Such a tree represents a Boolean function in the obvious way. We call the tree *trivial* if it contains exactly one nonleaf vertex. For instance, the majority function on 3 inputs can only be represented by a trivial tree.

A *tree decomposition* of f is a separation of f into the smallest possible components, with the exception of $(\neg)\,\mathrm{AND}_k$, $(\neg)\,\mathrm{OR}_k$, and $(\neg)\,\mathrm{XOR}_k$ components (where $(\neg)$ denotes possible negation), which are left intact. The choice of AND, OR, and XOR components is not arbitrary; these are precisely the three components that "associate," so that, for example, $\mathrm{AND}(x_1, \mathrm{AND}(x_2, x_3)) = \mathrm{AND}(\mathrm{AND}(x_1, x_2), x_3)$. Formally we have the following.

DEFINITION 6.2. *A* tree decomposition *of f is a distinct variable tree representing f such that*

(i) *no vertex is labeled with a function f that can be represented by a nontrivial tree, unless f is $(\neg)\,\mathrm{AND}_k$, $(\neg)\,\mathrm{OR}_k$, or $(\neg)\,\mathrm{XOR}_k$ for some k;*
(ii) *no vertex labeled with $(\neg)\,\mathrm{AND}_k$ has a child labeled with AND_l;*
(iii) *no vertex labeled with $(\neg)\,\mathrm{OR}_k$ has a child labeled with OR_l;*
(iv) *no vertex labeled with $(\neg)\,\mathrm{XOR}_k$ has a child labeled with $(\neg)\,\mathrm{XOR}_l$;*
(v) *any vertex labeled with a function that is constant on all but one input is labeled with $(\neg)\,\mathrm{AND}_k$ or $(\neg)\,\mathrm{OR}_k$.*

Let *double-negation* be the operation of negating the output of a function at some nonroot vertex v, then negating the corresponding input of the function at v's parent. Double-negation is a trivial way to obtain distinct decompositions. This caveat aside, we can assert uniqueness as follows.

THEOREM 6.3. *Every Boolean function has a unique tree decomposition, up to double-negation.*

We will build up to this uniqueness theorem via a sequence of preliminary results. Given a vertex v of a distinct variable tree, let $L(v)$ be the set of variables in the subtree of which v is the root. Assume that f is represented by two distinct tree decompositions, S and T, such that S has a vertex v_S and T has a vertex v_T with $L(v_S)$ and $L(v_T)$ incomparable (i.e., they intersect, but neither contains the other). We partition V_f into four sets of variables as follows: $A = L(v_S) - L(v_T)$, $B = L(v_T) - L(v_S)$, $I = L(v_S) \cap L(v_T)$, and $U = V_f - L(v_S) - L(v_T)$. Our strategy will be to derive increasingly strong constraints on how S and T can combine information

from A, B, I, and U. We do this by repeatedly restricting variables—considering f as, say, a function of I only—and then exploiting the fact that S and T must produce the same output, even though information travels along different routes in the two trees. Ultimately (in Lemma 6.5) we show that f is a function of $s(A)$, $r(I)$, and $t(B)$ for some Boolean functions s, r, and t. The problem thereby reduces to which Boolean functions of *three* variables have nonunique decompositions—and we can check that the only possibilities, AND, OR, and XOR, are ruled out by the definition of a tree decomposition.

Call a set of variables *unifiable* if there exists a vertex v, in *any* decomposition of f, such that $L(v) = V$. The preceding results imply that no pair of unifiable sets V_S, V_T is incomparable (Lemma 6.6): either $V_S \cap V_T = \phi$, $V_S \subseteq V_T$, or $V_T \subseteq V_S$. From there, it is readily shown that any decomposition must contain a vertex v with $L(v) = V$ for *every* unifiable V, from which the uniqueness theorem follows.

A remark on notation: we use subscripts to name Boolean functions (i.e., s_0, s_1, etc.) in order of their appearance, and superscripts to list which of A, B, I, and U are currently being restricted.

LEMMA 6.4. *There exist Boolean functions r, t_0^0, and t_0^1 such that f is a function of A, $r(I)$, $t_0^{r(I)}(B)$, and U.*

Proof. For any restriction u of U, we can write the output of S as $S^u[s_1(A, I), B]$, where S^u and s_1 are Boolean functions. Similarly we can write the output of T as $T^u[A, t_1(I, B)]$. We have that, for all settings of U,

$$S^u[s_1(A, I), B] = T^u[A, t_1(I, B)].$$

Consider a restriction b of B. This yields

$$S^{u,b}[s_1(A, I)] = T^u[A, t_2^b(I)]$$

for some Boolean function t_2^b. Therefore, for each b, s_1 depends on only a single bit obtained from I, namely, $t_2^b(I)$. Thus we can write $s_1(A, I)$ as $s_3(A, t_2^b(I))$ for some Boolean function s_3—or, even more strongly, as $s_3(A, s_4(I))$, since we know that s_1 does not depend on B. By analogous reasoning we can write $t_1(I, B)$ as $t_3(t_4(I), B)$ for some functions t_3 and t_4. Thus we have

$$S^u[s_3(A, s_4(I)), B] = T^u[A, t_3(t_4(I), B)].$$

Next we apply the restrictions $A = a$ and $B = b$, obtaining

$$S^{u,b}[s_3^a(s_4(I))] = T^{u,a}[t_3^b(t_4(I))],$$

which implies that, for some functions s_5 and t_5,

$$s_5(s_4(I)) = t_5(t_4(I))$$

for all I. This shows that $s_4(I)$ and $t_4(I)$ are equivalent up to negation of output, since S and T must depend on I for some restriction of A and B. Thus we have

$$(*) \qquad S^u[s_0^{r(I)}(A), B] = T^u[A, t_0^{r(I)}(B)]$$

for some Boolean functions $r(I)$, s_5^i, and t_5^i ($r \in \{0, 1\}$). ∎

We will henceforth think of $r(I)$ as a single Boolean variable.

LEMMA 6.5. *There exist Boolean functions s and t such that f is a function of $s(A)$, $r(I)$, $t(B)$, and U.*

Proof. Starting from (*), we next apply the restrictions $A = a$ and $r(I) = i$:

$$S^{u,a,i}[B] = T^{u,a}[t_0^i(B)].$$

Thus, for all restrictions of A and $r(I)$, S depends on only a single bit obtained from B, namely, $t_0^i(B)$. Note that t_0^i does not depend on A. Analogously, for both possible restrictions i of $r(I)$, T depends on only a single bit obtained from A, namely, $s_0^i(A)$. Thus we can write

$$s_6^u[s_0^i(A), t_0^i(B)] = t_6^u[s_0^i(A), t_0^i(B)],$$

where s_6^u and t_6^u are two-input Boolean functions. We claim that $s_0^0 = s_0^1$ and $t_0^0 = t_0^1$.

There must exist a restriction u of U such that s_6^u depends on both s_0^i and t_0^i. Suppose there exists a restriction b of B such that $t_0^0(b) \neq t_0^1(b)$. Now, s_0^i must be a nonconstant function, so find a constant c such that $s_6^u[c, t_0^i(b)]$ depends on t_0^i, and choose restrictions $A = a$ and $r(I) = i$ such that $s_0^i(a) = c$. (If s_6^u is an XOR function, then either $c = 0$ or $c = 1$ will work, whereas if s_6^u is an AND or OR function, then only one value of c will work.) For s_6^u to be well defined, we need that whenever $s_0^i(a) = c$, the value of i is determined, since

$$s_6^u[s_0^i(A), t_0^i(B)] = S^u[s_0^i(A), B]$$

and so the only access that s_6^u has to i is through s_0^i. This implies that s_0^i has the form $s(A) \wedge i$ or $s(A) \wedge \neg i$ for some function s. Therefore s_6^u can be written as $s_7^u[s(A), i, t_0^i(B)]$ for some function s_7^u. Now repeat the argument for t_6^u. We obtain that t_6^u can be written as $t_7^u[s_0^i(A), i, t(B)]$ for some functions t_7^u and t. Therefore

$$s_7^u[s(A), i, t_0^i(B)] = t_7^u[s_0^i(A), i, t(B)].$$

Thus we can take $t_0^i(B) = t(B)$ and $s_0^i(A) = s(A)$ and write s_7^u (as well as t_7^u) as $s_7^u[s(A), r(I), t(B)]$. □

Recall that a set $V \subseteq V_f$ is *unifiable* if there exists a vertex v, in some decomposition of f, such that $L(v) = V$.

LEMMA 6.6. *If V_S and V_T are unifiable, then either $V_S \cap V_T = \phi$, $V_S \subseteq V_T$, or $V_T \subseteq V_S$.*

Proof. Let $V_S = L(v_S)$ in decomposition S, and $V_T = L(v_T)$ in decomposition T, and suppose that V_S and V_T are incomparable. Let g_S be the function at v_S, and g_T the function at v_T. Defining A, I, B, and U as before, from Lemma 6.5 there exist Boolean functions $s(A)$, $r(I)$, and $t(B)$ such that

$$g_S = h_S(s(A), r(I)),$$
$$g_T = h_T(r(I), t(B))$$

for some two-variable Boolean h_S and h_T. Also, there exists a restriction $U = u$ for which f depends on all three of $s(A)$, $r(I)$, and $t(B)$. So the question reduces to which Boolean $\eta(a, i, b)$'s dependent on all three inputs are *associative*, in the sense that there exist Boolean η_1, η_2 and h_S, h_T for which

$$\eta(a, i, b) = \eta_1(h_S(a, i), b) = \eta_2(a, h_T(i, b)).$$

It is easily checked that the only possibilities are

$$(\neg)\,\mathrm{XOR}(a,i,b) \quad \text{or} \quad (\neg)\,\mathrm{AND}((\neg)a,(\neg)i,(\neg)b),$$

where $(\neg)$ denotes possible negation. Furthermore, η is determined up to negation, given h_S and h_T, so η cannot depend on u. In both the XOR and the AND (or equivalently OR) cases, v_S and v_T would have been collapsed to a single vertex in both S and T, by properties (ii)–(v) of a tree decomposition. Thus we have a contradiction. □

Now that we have ruled out the possibility of incomparable subtrees, we can establish uniqueness.

Proof of Theorem 6.3. It remains only to show that any decomposition must contain a vertex v with $L(v) = V$ for each unifiable V. Suppose that V is not represented in some decomposition F. Certainly $V \neq V_f$, so let V_P be the *parent set* of V in F: that is, the unique minimal set such that $V \subset V_P$ and there exists a vertex v_P in F with $L(v_P) = V_P$. Then the function at v_P is represented by a nontrivial tree containing a vertex v with $L(v) = V$. Were it not so represented, then for any Boolean function g on V there would exist a setting W of $V_P - V$ such that W, together with $g(V)$, would not suffice to determine the function h at v_P. Since f depends on h for some setting of $V_f - V_P$, it follows that v could not be a vertex in any decomposition. Furthermore, the function at v_P cannot be $(\neg)\,\mathrm{AND}_k$, $(\neg)\,\mathrm{OR}_k$, or $(\neg)\,\mathrm{XOR}_k$. If it were, then again v could not be a vertex in any decomposition, since it would need to be labeled correspondingly with $(\neg)\,\mathrm{AND}_k$, $(\neg)\,\mathrm{OR}_k$, or $(\neg)\,\mathrm{XOR}_k$. Having determined the unique set of vertices that comprise any decomposition, the vertices' labels are also determined up to double-negation. □

We now consider algorithms for finding the tree decomposition. First, given a subset G of V_f, there is a linear-time algorithm for deciding whether a Boolean function tree representing f could have a vertex u with $L(u) = G$. Consider the set F of $2^{n-|G|}$ restrictions on G induced by setting all the variables in $V_f - G$ to constant values. A vertex could have $L(u) = G$ if and only if all restrictions in F are identical up to negation, omitting constant functions. This can be checked in $O(N)$ time, which leads to an $O(N^2)$ algorithm for finding all vertices in the tree decomposition. (As a postprocessing step, the algorithm prunes superfluous AND_k, OR_k, and XOR_k vertices.)

However, we can reduce the running time to $O(N^{\log_2 3} \log N)$ by being more careful about how we check whether all restrictions in F are identical. The idea is to represent each restriction by a concise *code number*, which takes up only $O(n)$ bits rather than $2^{|G|}$ bits. We create the code numbers recursively, starting with the smallest restrictions and working up to larger ones. The code numbers need to satisfy the following conditions:

1. Two restrictions S and T over the same set of variables get mapped to identical code numbers if and only if $S = T$.
2. If S is constant or S is the negation of T, then these facts are easy to tell, given the code numbers of S and T.

We can satisfy these conditions by building up a binary tree of restrictions at each recursive call, then assigning each restriction a code number based on its position in the tree: 1 if it is the leftmost leaf, 2 if it is the second-to-leftmost, and so on. There are two exceptions: the constant 0 and 1 restrictions are assigned special code numbers Φ_0 and Φ_1, respectively, and if the negation of S was already assigned code number J, then S is assigned code number $-J$. For all $G \neq \phi$, each object inserted into the tree is two code numbers of size $|G| - 1$ restrictions concatenated together. Because this pair

of code numbers is then "hashed down" to a single number based on its position in the tree, the numbers always remain of size $O(n)$. In the pseudocode, B is the binary tree, $J[S]$ is the codeword of restriction S, and the operation $\odot$ denotes concatenation. The set VERTICES stores the final result, namely, all sets $H \subseteq V_f$ such that there is a vertex u in the decomposition of f having $L(u) = H$. After the main loop of the algorithm, a postprocessing step deletes redundant AND_k, OR_k, and $(\neg)\,\mathrm{XOR}_k$ vertices. This step looks for vertices u and v with $L(u)$ and $L(v)$ incomparable, which by Theorem 6.3 can only have arisen by AND_k, OR_k, or $(\neg)\,\mathrm{XOR}_k$.

ALGORITHM 6 (decomposes a Boolean function).

For all $G \subseteq V_f$ (in lexicographic order, starting with $G = \phi$):

1. *Initialize an empty self-balancing binary tree B.*
2. *Let Z be the set of all restrictions $S \in \{0, 1, *\}^n$ that fix exactly those variables not in G. Also, if $G \neq \phi$, then let k be the minimum i such that $v_i \in G$.*
3. *For all $S \in Z$,*
   - *If $G = \phi$, then insert $\Phi_{f(S)}$ into B. Otherwise, let S_0 and S_1 be further restrictions of S that fix v_k to 0 and 1, respectively, and insert $J[S_0] \odot J[S_1]$ into B.*
4. *For all $S \in Z$, assign S a code number $J[S]$ as follows:*
   - *For $i \in \{0, 1\}$, if $J[S_0] = J[S_1] = \Phi_i$, then $J[S] = \Phi_i$ also.*
   - *Otherwise, if $(-J[S_0]) \odot (-J[S_1])$ (corresponding to the negation $\neg S$ of S) is to the left of $J[S_0] \odot J[S_1]$ in B, then $J[S] = -J[\neg S]$.*
   - *Otherwise, if $J[S_0] \odot J[S_1]$ is the tth leaf of B in left-to-right order, then $J[S] = t$.*
5. *If $|G| \geq 2$ and, for all $S \in Z$, $|J[S]|$ is identical (omitting those S for which $J[S] = \Phi_0$ or $J[S] = \Phi_1$), then add G to* VERTICES*; otherwise do not.*

For each $i \in \{1, \ldots, n\}$, find all $G \in$ VERTICES *such that $v_i \in G$. Attempt to sort them into an ascending sequence $G_1 \subset G_2 \subset \cdots$. If a G_i to be inserted is incomparable with some G_j in the sequence, then leave G_j in the sequence, do not insert G_i, and flag both G_i and G_j for removal.*

Both the main loop and the second loop effectively perform an $O(\log N)$-time operation for all subsets of subsets of V_f. Therefore the total running time is $O(N^{\log_2 3} \log N)$.

7. Quantum query complexity. The quantum query complexity of a Boolean function f is the minimum number of oracle queries needed by a quantum computer to evaluate f. Here we are concerned only with the bounded-error query complexity $\mathrm{Q}_2(f)$ (defined in [7]), since approximating unitary matrices with finite precision introduces bounded error into any quantum algorithm. A quantum query algorithm Γ proceeds by an alternating sequence of $T + 1$ unitary transformations and T query transformations: $U_0 \to Q_1 \to U_1 \to \cdots \to Q_T \to U_{T+1}$. Then $\mathrm{Q}_2(f)$ is the minimum of T over all Γ that compute f with bounded error.

As mentioned in section 1, Barnum, Saks, and Szegedy [5] have recently obtained a polynomial-time algorithm to approximate $\mathrm{Q}_2(f)$. Here we show a weaker result: that if we limit the number of qubits, we can obtain a subexponential-time approximation algorithm via careful exhaustive search.

7.1. Overview of result. For what follows, it will be convenient to extend the quantum oracle model to allow intermediate observations. With an unlimited workspace, this cannot decrease the number of queries needed [8]. In the space-bounded setting, however, it might make a larger difference.

We define a *composite algorithm* Γ' to be an alternating sequence $\Gamma_1 \to D_1 \to \cdots \to \Gamma_t \to D_t$. Each Γ_i is a quantum query algorithm that uses T_i queries and at most m qubits of memory for some $m \geq \log_2 n + 2$. When Γ_i terminates, a basis state $|\psi_i\rangle$ is observed. Each D_i is a *decision point*, which takes as input the sequence $|\psi_1\rangle, \ldots, |\psi_i\rangle$, and as output decides whether to (1) halt and return $f = 0$, (2) halt and return $f = 1$, or (3) continue to Γ_{i+1}. (The final decision point, D_t, must choose between (1) and (2).) There are no computational restrictions placed on the decision points. However, a decision point cannot modify the quantum algorithms that come later in the sequence; it can only decide whether to continue with the sequence. For a particular input, let p_k be the probability, over all runs of Γ', that quantum algorithm Γ_k is invoked. Then Γ' uses a total number of queries $\sum_{k=1}^{t} p_k T_k$.

We define the space-bounded quantum query complexity $\mathrm{SQ}_{2,m}(f)$ to be the minimum number of queries used by any composite algorithm that computes f with error probability at most $1/3$ and that is restricted to m qubits. We give an approximation algorithm for $\mathrm{SQ}_{2,m}(f)$ taking time $2^{O(4^m mn)}$, which when $m = O(\log n)$ is $O(N^{\mathrm{polylog}(N)})$. The approximation ratio is $\sqrt{22}/3 + \epsilon$ for any $\epsilon > 0$. The difficulty in proving the result is as follows.

A unitary transformation is represented by a continuous-valued matrix, which might suggest that the quantum model of computation is analog rather than digital. But Bernstein and Vazirani [8] showed that, for a quantum computation taking T steps, the matrix entries need to be accurate only to within $O(\log T)$ bits of precision in the bounded-error model. However, when we try represent unitary transformations on a computer with finite precision, a new problem arises. On the one hand, if we allow only matrices that are exactly unitary, we may not be able to approximate every unitary matrix. So we also need to admit matrices that are *almost* unitary. For example, we might admit a matrix if the norm of each row is sufficiently close to 1, and if the inner product of each pair of distinct rows is sufficiently close to 0. But how do we know that every such matrix is close to some actual unitary matrix? If it is not, then the transformation it represents cannot even approximately be realized by a quantum computer.

We resolve this issue as follows. First, we show that every almost-unitary matrix is close to some unitary matrix in a standard metric. Second, we show that every unitary matrix is close to some almost-unitary matrix representable with limited precision. Third, we upper-bound the precision that suffices for a quantum algorithm, given a fixed accuracy that the algorithm needs to attain.

An alternative approach to approximating $\mathrm{SQ}_{2,m}(f)$ would be to represent each unitary matrix as a product of elementary gates. Kitaev [17] and independently Solovay [24] showed that a $2^m \times 2^m$ unitary matrix can be represented with arbitrary accuracy $\delta > 0$ by a product of $2^{O(m)\,\mathrm{polylog}(1/\delta)}$ unitary gates. But this yields a $2^{2^{O(m)\,\mathrm{polylog}(mn)}}$ algorithm, which is slower than ours. Perhaps the construction or its analysis can be improved; in any case, though, this approach is less natural for the setting of query complexity.

7.2. Almost-unitary matrices. Let $u \bullet v$ denote the conjugate inner product of u and v. The distance $|A - B|$ between matrices $A = (a_{ij})$ and $B = (b_{ij})$ in the $L_{\max}$ norm is defined to be $\max_{i,j} |a_{ij} - b_{ij}|$.

DEFINITION 7.1. *A matrix A is* q-almost-unitary *if* $|I - AA^\dagger| < q$.

In the following lemma, we start with an almost-unitary matrix A and construct an actual unitary matrix U that is close to A in the $L_{\max}$ norm.

LEMMA 7.2. *Let A be a q-almost-unitary $s \times s$ matrix, with $s \geq 2$ and $q \leq 1/4s$. Then there exists a unitary matrix U such that $|A - U| < 4.91q\sqrt{s}$.*

Proof. We first normalize each row A_i so that $A_i \bullet A_i = 1$. For each entry a_{ij},

$$|a_{ij}/(A_i \bullet A_i) - a_{ij}| = |a_{ij}||1 - (A_i \bullet A_i)|/|A_i \bullet A_i| < q(1+q)/(1-q).$$

We next form a unitary matrix B from A by using the classical Gram–Schmidt (CGS) orthogonalization procedure. The idea is to project A_2 to make it orthogonal to A_1, then project A_3 to make it orthogonal to both A_1 and A_2, and so on. Initially we set $B_1 \leftarrow A_1$. Then for each $2 \leq i \leq s$ we set $B_i \leftarrow A_i - \sum_{j=1}^{i-1}(A_i \bullet B_j)B_j$. Therefore

$$A_i \bullet B_k = (A_i \bullet A_k) - \sum_{j=1}^{k-1}(A_i \bullet B_j)(A_k \bullet B_j).$$

We need to show that the discrepancy between A and B does not increase too drastically as the recursion proceeds. Let $\sigma_k = \max_i A_i \bullet B_k$. By hypothesis, $\sigma_1 < q$. Then $\sigma_k \leq \sigma_1 + \sum_{j=1}^{k-1}\sigma_j^2$. Assume that $\sigma_k < q + 4q^2 s$ for all $k \leq K$. By induction,

$$\sigma_{K+1} < q + K(q + 4q^2 s)^2 \leq q + 4q^2 s$$

since $q \leq 1/4s$ and $K \leq s$. So for all k, $\sigma_k < q + 4q^2 s$.

Let $\phi = |A - B|$. By the definition of B, $\phi \leq \sigma_1|w_1| + \cdots + \sigma_s|w_s|$, where w is a column of B. Since $|w_1|^2 + \cdots + |w_s|^2 = 1$, ϕ is maximized when $w_i = \sigma_i\sqrt{s}/(\sigma_1 + \cdots + \sigma_s)$, or

$$\phi \leq \sigma_1^2 + \cdots + \sigma_s^2\sqrt{s}/(\sigma_1 + \cdots + \sigma_s) \leq (q + 4q^2 s)^2\sqrt{s}/q.$$

Adding $q(1+q)/(1-q)$ from normalization yields a quantity less than $(4 + 9\sqrt{2}/14)q\sqrt{s} \approx 4.91q\sqrt{s}$. This can be seen by working out the arithmetic for the worst case of $s = 2$, $q = 1/4s$. □

The next lemma, which is similar to Lemma 6.1.3 of [8], is a sort of converse to Lemma 7.2: we start with an arbitrary unitary matrix and show that truncating its entries to a precision $\delta > 0$ produces an almost-unitary matrix.

LEMMA 7.3. *Let U and V be $s \times s$ matrices with $s \geq 2$ and $|U - V| < \delta$. If U is unitary, then V is $(2\delta\sqrt{s} + \delta^2 s)$-almost-unitary.*

Proof. First,

$$U_i \bullet U_i = \sum_{k=1}^{s} |u_k + \gamma_k|^2 = 1 + \sum_{k=1}^{s}(u_k\gamma_k^* + u_k^*\gamma_k + \gamma_k\gamma_k^*),$$

where the u_k's are entries of U and the γ_k's are error terms satisfying $|\gamma_k| < \delta$. Thus, by the Cauchy–Schwarz inequality, $U_i \bullet U_i$ differs from 1 by at most $2\delta\sqrt{s} + \delta^2 s$. Second, for $i \neq j$,

$$U_i \bullet U_j = \sum_{k=1}^{s}(u_k + \gamma_k)(u_k + \eta_k)^*,$$

where the γ_k's and η_k's are error terms, and the argument proceeds analogously. □

7.3. Searching for quantum algorithms. In this section we use the results on almost-unitary matrices to construct an algorithm. First we need a lemma about error buildup in quantum algorithms, which is similar to Corollary 3.4.4 of [8] (though the proof technique is different).

LEMMA 7.4. *Let $U_1, \ldots, U_T$ be $s \times s$ unitary matrices, $\widehat{U_1}, \ldots, \widehat{U_T}$ be $s \times s$ arbitrary matrices, and v be an $s \times 1$ vector with $\|v\|_2 = 1$. Suppose that, for all i, $|\widehat{U_i} - U_i| < 1/cs$, where $c > T/2$. Then $\widehat{U_1} \cdots \widehat{U_T} v$ differs from $U_1 \cdots U_T v$ by at most $2T/[\sqrt{s}(2c - T)]$ in the L_2 norm.*

Proof. For each i, let $E_i = \widehat{U_i} - U_i$. By hypothesis, every entry of E_i has magnitude at most $1/cs$; thus, each row or column w of E_i has $\|w\|_2 \leq 1/(c\sqrt{s})$. Then

$$\widehat{U_1} \cdots \widehat{U_T} v = (U_1 + E_1) \cdots (U_T + E_T) v.$$

The right-hand side, when expanded, has 2^T terms. Any term containing k matrices E_i has L_2 norm at most $s^{-1/2} c^{-k}$ and can therefore add at most $c^{-k}/\sqrt{s}$ to the discrepancy with $U_1 \cdots U_T v$. Thus the total discrepancy is at most

$$s^{-1/2} \sum_{k=1}^{T} \binom{T}{k} (1/c)^k < s^{-1/2} (e^{T/c} - 1).$$

Since $d \ln t / dt$ evaluated at $t = 2c$ is $1/2c$ and since $\ln t$ is concave,

$$\ln(2c + T) - \ln(2c - T) \geq 2T/2c = T/c$$

when $T < 2c$. Therefore $e^{T/c} \leq (2c + T)/(2c - T)$, and the discrepancy is at most $2T/[\sqrt{s}(2c - T)]$ in the L_2 norm. □

Applying Lemmas 7.2, 7.3, and 7.4, we now prove the main theorem.

THEOREM 7.5. *There exists an approximation algorithm for $\mathrm{SQ}_{2,m}(f)$ taking time $2^{O(4^m mn)}$, with approximation ratio $\sqrt{22}/3 + \epsilon$.*

Proof. Given f, we want, subject to the following two constraints, to find an algorithm Γ that approximates f with a minimum number of queries. First, Γ uses at most m qubits, meaning that $s = 2^m$ and the relevant matrices are $2^m \times 2^m$. Second, the correctness probability of Γ is known to a constant accuracy $\pm\varepsilon$. Certainly the number T of queries never needs to be more than n, for, although each quantum algorithm is space-bounded, the *composite* algorithm need not be. Let λ be the $L_{\max}$ error we can tolerate in the matrices, and let Δ be the resultant L_2 error in the final states. Setting $c = 1/(\lambda 2^m)$, by Lemma 7.4 we have

$$\Delta \leq 2n/[2^{m/2}(2^{1-m}/\lambda - n)].$$

From the Cauchy–Schwarz inequality, one can show that $\varepsilon \leq 2\Delta$. Then solving for $1/\lambda$, $1/\lambda \leq 2^{m/2} n(2/\varepsilon + 1)$, which, since ε is constant, is $O(2^{m/2} n)$. Solving for c, we can verify that $c > T/2$, as required by Lemma 7.4. If we generate almost-unitary matrices, they need to be within λ of actual unitary matrices. By Lemma 7.2 we can use $\lambda/(4.91\sqrt{s})$-almost-unitary matrices. Finally we need to ensure that we approximate every unitary matrix. Let δ be the needed precision. Invoking Lemma 7.3, we set $\lambda/(4.91\sqrt{s}) \geq 2\delta\sqrt{s} + \delta^2 s$ and obtain that

$$\delta \leq \max[\lambda/(9.82s), \lambda^{1/2}/(2.22 s^{3/4})]$$

is sufficient.

Therefore the number of bits of precision needed per entry, $\log(1/\delta)$, is $O(m)$. We thus need only $O(4^m mn)$ bits to specify Γ, and can search through all possible Γ in time $2^{O(4^m mn)}$. The amount of time needed to evaluate a composite algorithm Γ' is polynomial in m and n and is absorbed into the exponent. The approximation algorithm is this: first let $\varepsilon > 0$ be a constant at most 0.0268, and let $\omega = \frac{22}{9} + \frac{4}{3}\varepsilon - 8\varepsilon^2$. Then find the smallest T such that the maximum probability of correctness over all T-query algorithms Γ' is at least $2/3 - \varepsilon$ (subject to $\pm\varepsilon$ uncertainty), and return $T\sqrt{\omega}$. The algorithm achieves an approximation ratio of $\sqrt{\omega}$, for the following reasons. First, $T \leq \mathrm{SQ}_{2,m}(f)$. Second, $\omega T \geq \mathrm{SQ}_{2,m}(f)$, since by repeating the optimal algorithm Γ^* until it returns the same answer twice (which takes either two or three repetitions), the correctness probability can be boosted above $2/3$. Finally, a simple calculation reveals that Γ^* returns the same answer twice after expected number of invocations ω. ∎

8. Open problems. Implicit in the paper of Ambainis [3] is a novel Boolean function property, which is used to obtain lower bounds on quantum query complexity. To take a special case, given a function f and a set S of inputs, let the "Ambainis density" $\mathrm{AD}_S(f)$ be the minimum, over all $X \in S$, of the number of i such that $X^{(i)} \in S$ and $f(X) \neq f(X^{(i)})$. (Here $X^{(i)}$ denotes X with the ith bit negated.) Then let $\mathrm{AD}(f)$ be the maximum of $\mathrm{AD}_S(f)$ over all S. Ambainis shows that $\mathrm{Q}_2(f) = \Omega(\mathrm{AD}(f))$. How efficient an algorithm can we find for $\mathrm{AD}(f)$? Additionally, Bar-Yossef, Kumar, and Sivakumar [6] have defined "approximate" versions of measures such as block sensitivity. Can we extend the algorithms given in this paper to compute those measures?

Acknowledgments. I thank Rob Pike and Lorenz Huelsbergen for sponsoring the internship during which this work was done and for helpful discussions; Andris Ambainis, Wim van Dam, Michael Saks, Umesh Vazirani, and the anonymous reviewers for their comments; Ronald de Wolf for discussions of space-bounded quantum query complexity; and Peter Bro Miltersen for correspondence.

REFERENCES

[1] S. Aaronson, *Quantum certificate complexity*, in Proceedings of the 18th Annual IEEE Conference on Computational Complexity, Aarhus, Denmark, 2003, IEEE Press, Los Alamitos, CA, 2003, pp. 171–178.

[2] S. Aaronson, *Quantum lower bound for the collision problem*, in Proceedings of the 34th ACM Symposium on Theory of Computing (STOC), Montreal, 2002, ACM, New York, 2002, pp. 635–642.

[3] A. Ambainis, *Quantum lower bounds by quantum arguments*, J. Comput. System Sci., 64 (2002), pp. 750–767.

[4] A. Ambainis and R. de Wolf, *Average-case quantum query complexity*, in Proceedings of the Symposium on Theoretical Aspects of Computing Science, 2000.

[5] H. Barnum, M. Saks, and M. Szegedy, *Quantum decision trees and semidefinite programming*, in Proceedings of the 18th Annual IEEE Conference on Computational Complexity, Aarhus, Denmark, 2003, IEEE Press, Los Alamitos, CA, 2003, pp. 179–193.

[6] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, *Sampling algorithms: Lower bounds and applications*, in Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC), Crete, 2001, ACM, New York, 2001, pp. 266–275.

[7] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf, *Quantum lower bounds by polynomials*, J. ACM, 48 (2001), pp. 778–797.

[8] E. Bernstein and U. Vazirani, *Quantum complexity theory*, SIAM J. Comput., 26 (1997), pp. 1411–1473.

[9] H. Buhrman and R. de Wolf, *Complexity measures and decision tree complexity: A survey*, Theoret. Comput. Sci., 288 (2002), pp. 21–43.

[10] S. L. A. Czort, *The Complexity of Minimizing Disjunctive Normal Form Formulas*, Master's thesis, University of Aarhus, Aarhus, Denmark, 1999.
[11] W. van Dam, *Quantum oracle interrogation: Getting all the information for almost half the price*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS), Palo Alto, CA, 1998, IEEE, Piscataway, NJ, 1998, pp. 362–367.
[12] D. Guijarro, V. Lavín, and V. Raghavan, *Exact learning when irrelevant variables abound*, Inform. Process. Lett., 70 (1999), pp. 233–239.
[13] L. K. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the 28th ACM Symposium on Theory of Computing (STOC), Philadelphia, 1996, ACM, New York, 1996, pp. 212–219.
[14] T. Hancock, T. Jiang, M. Li, and J. Tromp, *Lower bounds on learning decision lists and trees*, Inform. and Comput., 126 (1996), pp. 114–122.
[15] K. Hoffman and R. Kunze, *Linear Algebra*, Prentice–Hall, Englewood Cliffs, NJ, 1971.
[16] L. Hyafil and R. L. Rivest, *Constructing optimal binary decision trees is NP-complete*, Inform. Process. Lett., 5 (1976), pp. 15–17.
[17] A. Yu. Kitaev, *Quantum computations: Algorithms and error correction*, Russian Math. Surveys, 52 (1997), pp. 1191–1249.
[18] N. Nisan, *CREW PRAMs and decision trees*, SIAM J. Comput., 20 (1991), pp. 999–1007.
[19] N. Nisan and M. Szegedy, *On the degree of Boolean functions as real polynomials*, Comput. Complexity, 4 (1994), pp. 301–313.
[20] A. A. Razborov and S. Rudich, *Natural proofs*, J. Comput. System Sci., 55 (1997), pp. 24–35.
[21] M. Saks and A. Wigderson, *Probabilistic Boolean decision trees and the complexity of evaluating game trees*, in Proceedings of the 27th IEEE Symposium on Foundations of Computer Science (FOCS), IEEE, Piscataway, NJ, 1986, pp. 29–38.
[22] M. Santha, *On the Monte Carlo decision tree complexity of read-once formulae*, Random Structures Algorithms, 6 (1995), pp. 75–87.
[23] D. Sieling, *Minimization of decision trees is hard to approximate*, in Proceedings of the 18th Annual IEEE Conference on Computational Complexity, Aarhus, Denmark, 2003, IEEE Press, Los Alamitos, CA, 2003, pp. 84–92.
[24] R. Solovay, *Lie groups and quantum circuits*, talk presented at the Workshop on Mathematics of Quantum Computation, Mathematical Sciences Research Institute, Berkeley, CA, 2000.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1158–1184

ON TESTING CONVEXITY AND SUBMODULARITY*

MICHAL PARNAS†, DANA RON‡, AND RONITT RUBINFELD§

Abstract. Convex and submodular functions play an important role in many applications, and in particular in combinatorial optimization. Here we study two special cases: convexity in one dimension and submodularity in two dimensions. The latter type of functions are equivalent to the well-known *Monge matrices*. A matrix $V = \{v_{i,j}\}_{i,j=0}^{i=n_1,j=n_2}$ is called a Monge matrix if for every $0 \le i < i' \le n_1$ and $0 \le j < j' \le n_2$ we have $v_{i,j} + v_{i',j'} \le v_{i,j'} + v_{i',j}$. If inequality holds in the opposite direction, then V is an *inverse Monge* matrix (supermodular function). Many problems, such as the traveling salesperson problem and various transportation problems, can be solved more efficiently if the input is a Monge matrix.

In this work we present testing algorithms for the above properties. A testing algorithm for a predetermined property $\mathcal{P}$ is given query access to an unknown function f and a distance parameter ϵ. The algorithm should accept f with high probability if it has the property $\mathcal{P}$ and reject it with high probability if more than an ϵ-fraction of the function values should be modified so that f obtains the property. Our algorithm for testing whether a 1-dimensional function $f : [n] \to \mathbb{R}$ is convex (concave) has query complexity and running time of $O((\log n)/\epsilon)$. Our algorithm for testing whether an $n_1 \times n_2$ matrix V is a Monge (inverse Monge) matrix has query complexity and running time of $O((\log n_1 \cdot \log n_2)/\epsilon)$.

Key words. property testing, convex functions, Monge matrices, approximation algorithms, randomized algorithms

AMS subject classifications. 68W40, 68W25, 68W20, 39B62

DOI. 10.1137/S0097539702414026

1. Introduction.

Convex functions and their combinatorial analogues, submodular functions, play an important role in many disciplines and applications, including combinatorial optimization, game theory, probability theory, and electronic trade. Such functions exhibit a rich mathematical structure (see Lovász [Lov83]), which often makes it possible to efficiently find their minimum [GLS81, IFF01, Sch00] and thus leads to efficient algorithms for many important optimization problems. Convex functions over discrete domains are defined as follows.

DEFINITION 1 (convex and concave). *Let f be a function defined over a discrete domain X. The function f is* convex *if for all $x, y \in X$ and for all $0 \le \alpha \le 1$ such that $\alpha x + (1-\alpha)y \in X$, it holds that $f(\alpha x + (1-\alpha)y) \le \alpha f(x) + (1-\alpha)f(y)$. The function f is* concave *if for all $x, y \in X$ and for all $0 \le \alpha \le 1$ such that $\alpha x + (1-\alpha)y \in X$, it holds that $f(\alpha x + (1-\alpha)y) \ge \alpha f(x) + (1-\alpha)f(y)$.*

Submodular functions are defined as follows: Let $\mathcal{I} = I_1 \times I_2 \times \cdots \times I_d$, $d \ge 2$, be a product space where $I_q \subseteq \mathbb{R}$. In particular, we are interested in discrete domains $I_q = \{0, \ldots, n_q\}$. The *join* and *meet* operations are defined for every $x, y \in \mathcal{I}$ as follows:

$$(x_1, \ldots, x_d) \vee (y_1, \ldots, y_d) \stackrel{\text{def}}{=} (\max\{x_1, y_1\}, \ldots, \max\{x_d, y_d\})$$

*Received by the editors September 4, 2002; accepted for publication (in revised form) May 23, 2003; published electronically August 6, 2003. Part of this research was done while the first two authors were visiting NEC Research Institute.

http://www.siam.org/journals/sicomp/32-5/41402.html

†The Academic College of Tel-Aviv-Yaffo, 4 Antokolsky st., Tel-Aviv, Israel (michalp@mta.ac.il).

‡Department of EE – Systems, Tel-Aviv University, Ramat Aviv, Israel (danar@eng.tau.ac.il).
The research of this author was supported by Israel Science Foundation grant 32/00-1.
§NEC Research Institute, Princeton, NJ (ronitt@research.nj.nec.com).

and

$$(x_1, \ldots, x_d) \wedge (y_1, \ldots, y_d) \stackrel{\text{def}}{=} (\min\{x_1, y_1\}, \ldots, \min\{x_d, y_d\}),$$

respectively.

DEFINITION 2 (submodularity and supermodularity). *A function $f : \mathcal{I} \to \mathbb{R}$ is* submodular *if for every $x, y \in \mathcal{I}$, $f(x \vee y) + f(x \wedge y) \leq f(x) + f(y)$. The function f is* supermodular *if for every $x, y \in \mathcal{I}$, $f(x \vee y) + f(x \wedge y) \geq f(x) + f(y)$.*

Certain subclasses of submodular functions are of particular interest. One such subclass is that of *submodular set* functions, which are defined over binary domains. That is, $I_q = \{0, 1\}$ for every $1 \leq q \leq d$, and so each $x \in \mathcal{I}$ corresponds to a subset of $\{1, \ldots, d\}$. Such functions are used, for example, in the scenario of combinatorial auctions on the Internet (e.g., [dVV00, LLN01]).

Another important subclass is the class of *Monge* functions, which are obtained when the domain is large but the dimension is $d = 2$. Since such functions are 2-dimensional, it is convenient to represent them as 2-dimensional matrices, which are referred to as *Monge matrices*. When the function is a 2-dimensional supermodular function the corresponding matrix is called an *inverse Monge matrix*.

The first problem that was shown to be solvable more efficiently if the underlying cost matrix is a Monge matrix is the classical Hitchcock transportation problem (see Hoffman [Hof63]). Since then it has been shown that many other combinatorial optimization problems can be solved more efficiently in this case (e.g., weighted bipartite matching and NP-hard problems such as the traveling salesperson problem). See [BKR96] for a comprehensive survey on Monge matrices and their applications.

1.1. Testing convexity and submodularity. In this paper we approach the questions of convexity and submodularity from within the framework of property testing [RS96, GGR98]. (For surveys on property testing see [Ron01, Fis01].) Let f be a fixed but unknown function, and let $\mathcal{P}$ be a fixed property of functions (such as the convexity or submodularity of a function). A testing algorithm for the property $\mathcal{P}$ should determine, by querying f, whether f has the property $\mathcal{P}$ or whether it is ϵ-*far* from having the property for a given distance parameter ϵ. By ϵ-*far* we mean that more than an ϵ-fraction of the values of f should be modified so that f obtains the desired property $\mathcal{P}$.

Our results. We present efficient testing algorithms for discrete convexity in one dimension and for Monge matrices. Specifically, we do the following:

- We describe and analyze an algorithm that tests whether a function $f : [n] \to \mathbb{R}$ is convex (concave). The running time of this algorithm is $O(\log n/\epsilon)$.
- We describe and analyze a testing algorithm for Monge and inverse Monge matrices whose running time is $O((\log n_1 \cdot \log n_2)/\epsilon)$ when given an $n_1 \times n_2$ matrix.

  Furthermore, the testing algorithm for inverse Monge matrices can be used to derive a testing algorithm, with the same complexity, for an important subfamily of Monge matrices called *distribution matrices*. A matrix $V = \{v_{i,j}\}$ is said to be a distribution matrix if there exists a nonnegative *density matrix* $D = \{d_{i,j}\}$ such that every entry $v_{i,j}$ in V is of the form $v_{i,j} = \sum_{k \leq i} \sum_{\ell \leq j} d_{k,\ell}$. In other words, the entry $v_{i,j}$ corresponds to the cumulative density of all entries $d_{k,\ell}$ such that $k \leq i$ and $\ell \leq j$.

In both cases the complexity of the algorithms is linear in $1/\epsilon$ and polylogarithmic in the size of the domain.

1.2. Techniques.

Convexity in one dimension. We start with the following basic observation: A function $f : [n] \to \mathbb{R}$ is convex if and only if for every $1 \leq i \leq n-1$, $(f(i+1) - f(i)) - (f(i) - f(i-1)) \geq 0$. Given this characterization, consider the *difference* function f', which is defined as $f'(i) = f(i) - f(i-1)$. The function f' can be viewed as the discrete analogue of the first derivative of f. By the above observation we have that f is convex if an only if f' is monotone nondecreasing. Hence, a tempting approach for testing whether f is convex would be to test whether f' is monotone nondecreasing, where this can be done in time $O(\log n/\epsilon)$ [EKK+00, BRW99, DGL+99].

Unfortunately, this approach does not work. There are functions f that are very far from convex, but their difference function f' is very close to monotone.[1] Therefore, instead of considering only consecutive points $i, i+1$, we consider pairs of points $i, j \in [n]$ that are not necessarily consecutive. More precisely, we select intervals $\{i, \ldots, j\}$ of varying lengths and check that for each interval selected, certain constraints are satisfied. If f is convex, then these constraints are satisfied for every interval. On the other hand, we show that if f is ϵ-far from convex, then the probability that we observe a violation of some constraint is sufficiently large.

Monge matrices. As stated above, it is convenient to represent 2-dimensional submodular functions as 2-dimensional Monge matrices. Thus a function $f : \{0, \ldots, n_1\} \times \{0, \ldots, n_2\} \to \mathbb{R}$ can be represented as the matrix $V = \{v_{i,j}\}_{i,j=0}^{i=n_1,j=n_2}$, where $v_{i,j} = f(i,j)$. Observe that for every pair of indices $(i,j'), (i',j)$ such that $i < i'$ and $j < j'$ we have that $(i,j') \vee (i',j) = (i',j')$ and $(i,j') \wedge (i',j) = (i,j)$. It follows from Definition 2 that V is a Monge matrix (f is a 2-dimensional submodular function) if and only if

$$\forall i, j, i', j' \quad \text{s.t.} \quad i < i',\ j < j' \ : \qquad v_{i,j} + v_{i',j'} \leq v_{i,j'} + v_{i',j}$$

and V is an inverse Monge matrix (f is a 2-dimensional supermodular function) if and only if

$$\forall i, j, i', j' \quad \text{s.t.} \quad i < i',\ j < j' \ : \qquad v_{i,j} + v_{i',j'} \geq v_{i,j'} + v_{i',j}\ .$$

That is, in both cases we have a constraint for every quadruple $v_{i,j}$, $v_{i',j'}$, $v_{i,j'}$, $v_{i',j}$ such that $i < i'$ and $j < j'$.[2] Our algorithm selects such quadruples according to a particular (nonuniform) distribution and verifies that the constraint is satisfied for every quadruple selected. Clearly, the algorithm always accepts Monge matrices. The main thrust of the analysis is in showing that if the matrix V is far from being Monge, then the probability of obtaining a "bad" quadruple is sufficiently large.

A central building block in proving the above is the following combinatorial problem, which may be of independent interest. Let C be a given matrix, possibly containing negative values, and let R be a subset of positions in C. We are interested in refilling the entries of C that reside in R with *nonnegative* values such that the following constraint is satisfied: for every position (i,j) that does not belong to R, the sum of the modified values in C that are below[3] (i,j) is the same as in the original

[1] In particular, consider the function f such that for every $i \leq n/2$, $f(i) = i$, and for $i > n/2$, $f(i) = i - 1$. In other words, $f'(i) = 1$ for every i except $i = n/2$, where $f'(i) = 0$. Then f' is very close to monotone, but it is not hard to verify that f is far from convex.

[2] It is easy to verify that for all other i, j, i', j' (with the exception of the symmetric case where $i' < i$ and $j' < j$), the constraint holds trivially (with equality).

[3] We denote the lower left position of the matrix C by $(0,0)$.

matrix C. That is, the sum of the modified values in entries (k, ℓ) such that $k \leq i$ and $j \leq \ell$ remains as it was.

We provide sufficient conditions on C and R under which the above is possible and describe the corresponding procedure that refills the entries of C that reside in R. Our starting point is a simple special case in which R corresponds to a submatrix of C. In such a case it suffices that for each row and each column in R, the sum of the corresponding entries in the original matrix C is nonnegative. Under these conditions a simple greedy algorithm can modify C as required. Our procedure for general subsets R is more involved but uses the submatrix case as a subroutine.

1.3. Further research. We suggest the following open problems. First, it remains open to determine the complexity of testing discrete convexity (concavity) when the dimension d of the input domain is greater than 1 and for testing submodular (supermodular) functions when the dimension d is greater than 2. Note that though submodular functions can be viewed as a certain interpretation of convexity in dimensions $d \geq 2$, they do not necessarily satisfy Definition 1.

It seems that our algorithm for testing Monge matrices and its analysis can be extended to work for testing the special case of distribution matrices of dimension $d > 2$, where the complexity of the resulting algorithm is $O((\prod_{q=1}^{d} \log n_q)/\epsilon)$. However, as opposed to the $d = 2$ case, where Monge matrices are only slightly more general than distribution matrices, for $d > 2$ Monge matrices are more expressive. Hence it is not immediately clear how to adapt our algorithm to testing Monge matrices in higher dimensions.

It would also be interesting to find an efficient testing algorithm for the subclass of submodular set functions, which are defined over binary domains.

Finally, in many optimization problems it is enough that the underlying cost matrix is a permutation of a Monge matrix. In such cases it may be useful to test whether a given matrix is a permutation of some Monge matrix or far from any permuted Monge matrix.

Organization. The testing algorithm for convexity is described in section 2. The remainder of the paper is dedicated to testing Monge matrices. In section 3 we describe several building blocks that will be used by our testing algorithm for Monge matrices. In section 4 we describe a testing algorithm for Monge matrices whose complexity is $O(n/\epsilon)$, where we assume for simplicity that the matrix is $n \times n$. Building on this algorithm and its analysis, in section 5 we present a significantly faster algorithm whose complexity is $O\left((\log^2 n)/\epsilon\right)$. We conclude this section with a short discussion concerning distribution matrices.

2. Testing convexity in one dimension. As noted in the introduction, in the case that the domain is $X = [n] = \{0, \ldots, n\}$, we get the following characterization for convexity, whose proof is included for completeness.

CLAIM 1. *A function $f : [n] \to \mathbb{R}$ is* convex *if and only if for all $1 \leq i \leq n-1$, $f(i) - f(i-1) \leq f(i+1) - f(i)$.*

Proof. If f is convex, then in particular for $x = i - 1$, $y = i + 1$, and $\alpha = 1/2$ we have $\alpha x + (1-\alpha)y = \frac{i-1}{2} + \frac{i+1}{2} = i$. By Definition 1, $f(i) \leq \frac{1}{2}f(i-1) + \frac{1}{2}f(i+1)$, or, equivalently, $f(i) - f(i-1) \leq f(i+1) - f(i)$.

In the other direction, suppose that $f(i) - f(i-1) \leq f(i+1) - f(i)$ for every $1 \leq i \leq n-1$. Consider any $x, y \in [n]$ and $0 < \alpha < 1$ such that $z = \alpha \cdot x + (1-\alpha) \cdot y$ is an integer. Assume without loss of generality that $x < y$. Now we have that

$$f(y)-f(y-1) \geq f(y-1)-f(y-2) \geq \cdots \geq f(z+1)-f(z) \geq f(z)-f(z-1) \geq \cdots \geq f(x+1)-f(x).$$

Then, since the differences are monotone nonincreasing, the average of the first $\alpha(y-x)$ differences is greater than or equal to the average of the next $(1-\alpha)(y-x)$ differences. Since $z = y - \alpha(y-x) = x + (1-\alpha)(y-x)$, we have that

$$\frac{(f(y)-f(y-1))+(f(y-1)-f(y-2))+\cdots+(f(z+1)-f(z))}{\alpha(y-x)} \tag{1}$$

$$\geq \frac{(f(z)-f(z-1))+(f(z-1)-f(z-2))+\cdots+(f(x+1)-f(x))}{(1-\alpha)(y-x)}. \tag{2}$$

This is equivalent to $(1-\alpha)(f(y)-f(z)) \geq \alpha(f(z)-f(x))$; that is, $f(z) \leq \alpha f(x) + (1-\alpha)f(y)$ as required. □

Denote by $I_{i,j}$ the interval $\{i, i+1, \ldots, j\}$ of points. Let $mid = \lfloor (i+j)/2 \rfloor$ be the midpoint of $I_{i,j}$.

DEFINITION 3. *For every $0 \leq i < j \leq n$ such that $j - i > 7$, we say that the interval $I_{i,j}$ is* good with respect to f *if the following holds:*

$$f(i+1)-f(i) \leq \frac{f(mid-1)-f(i+1)}{(mid-1)-(i+1)} \leq f(mid)-f(mid-1) \leq f(mid+1)-f(mid)$$
$$\leq f(mid+2)-f(mid+1) \leq \frac{f(j-1)-f(mid+2)}{(j-1)-(mid+2)} \leq f(j)-f(j-1).$$

Otherwise, we say that the interval is bad with respect to f. *If $j - i \leq 7$, then $I_{i,j}$ is good with respect to f if and only if the function f is convex over $I_{i,j}$.*

In order to test if f is convex we test recursively if subintervals of $I_{0,n}$ are good.

ALGORITHM 1 (Test-Convex).

1. *Repeat $2/\epsilon$ times:* Test-Interval($I_{0,n}$).
2. *If all of the tests in step 1 accepted, then accept; otherwise, reject.*

PROCEDURE 1 (Test-Interval($I_{i,j}$)).

1. *Check that $I_{i,j}$ is good with respect to f. If not, reject.*
2. *If $j - i > 7$, then: Uniformly at random call either* Test-Interval($I_{i,mid}$) *or* Test-Interval($I_{mid+1,j}$), *where $mid = \lfloor (i+j)/2 \rfloor$.*
3. *If the test in step 2 accepted, then accept; otherwise, reject.*

THEOREM 1. *If f is convex, then Algorithm 1 always accepts, and if f is ϵ-far from convex, then the algorithm rejects with a probability of at least $2/3$.*

Proof. For the sake of brevity, unless stated otherwise, when we say that an interval is good, then we mean with respect to f. If f is convex, then all intervals $I_{i,j}$ are good, and hence Algorithm 1 accepts with probability 1. In order to prove that if f is ϵ-far from convex, then the algorithm rejects with probability of at least $2/3$, we prove the contrapositive statement. Assume that the algorithm accepts with a probability greater than $1/3$. We will show that f is ϵ-close to a convex function.

To this end we define a tree whose vertices correspond to all possible intervals $I_{i,j}$ that may be tested recursively in calls to Test-Interval($I_{i,j}$). Specifically, the root of the tree corresponds to $I_{0,n}$. The children of the internal vertex corresponding to $I_{i,j}$ are the vertices corresponding to $I_{i,mid}$ and $I_{mid+1,j}$, where $mid = \lfloor (i+j)/2 \rfloor$. The leaves of the tree correspond to the smallest intervals tested, that is, intervals $I_{i,j}$ for which $j - i \leq 7$.

We say that an internal vertex in the tree is good if the corresponding interval is good. We say that a leaf is good if its corresponding interval and all its ancestors are good. Otherwise, the vertex (leaf) is bad. We say that a path from the root to a leaf is good if all vertices along it are good. Otherwise, the path is bad. For each level ℓ in the tree, $\ell = 0, \ldots, \log n$, let $\mathcal{B}_\ell$ be the subset of vertices in the ℓth level of the

tree that are bad but whose ancestors are all good. Let $\mathcal{B} = \bigcup_\ell \mathcal{B}_\ell$, and let ϵ_ℓ be the fraction of vertices in level ℓ of the tree that belong to $\mathcal{B}_\ell$.

Subclaim 1. If Algorithm 1 accepts f with a probability greater than $1/3$, then $\sum_\ell \epsilon_\ell \leq \epsilon$.

Proof. Assume by contradiction that $\sum_\ell \epsilon_\ell > \epsilon$. Observe that by the definition of $\mathcal{B}$, all leaves which are descendents of a vertex in $\mathcal{B}$ are bad, and every bad leaf either belongs to $\mathcal{B}$ or has a single ancestor in $\mathcal{B}$. Therefore, if $\sum_\ell \epsilon_\ell > \epsilon$, then the fraction of bad leaves is greater than ϵ. But in such a case, the probability that the algorithm does not follow a bad path to a bad leaf (passing through a vertex in $\mathcal{B}$) in any one of its $2/\epsilon$ iterations is at most $(1-\epsilon)^{2/\epsilon} < e^{-2} < 1/3$. This contradicts our assumption that the algorithm accepts with a probability greater than $1/3$.

Hence we assume from now on that $\sum_\ell \epsilon_\ell \leq \epsilon$. Note also that in this case $I_{0,n} \notin \mathcal{B}$. We show how to modify f in at most $\epsilon \cdot n$ places so that the resulting function, denoted g, is convex. In particular, we shall modify the value of f on every bad interval $I_{i,j}$ whose corresponding vertex in the tree belongs to $\mathcal{B}$. The value of g is defined to be the same as the value of f on all points outside of these intervals. Since $\sum_\ell \epsilon_\ell \leq \epsilon$, the total fraction of points modified is at most ϵ as required. Observe that by the definition of the tree and $\mathcal{B}$, for every two intervals whose corresponding vertices belong to $\mathcal{B}$, the intersection of the intervals is empty. Hence we can modify each one of these intervals independently.

Let $I_{i,j}$ be a bad interval corresponding to a vertex in $\mathcal{B}$. We modify f on points in $I_{i,j}$ as follows:

- $f(i), f(i+1)$, $f(j-1)$, and $f(j)$ remain unchanged. That is, set $g(i) = f(i)$, $g(i+1) = f(i+1)$, $g(j-1) = f(j-1)$, and $g(j) = f(j)$.
- For every t, $i+1 < t < j-1$, set $g(t) = f(i+1) + \frac{f(j-1)-f(i+1)}{(j-1)-(i+1)} \cdot (t-(i+1))$.

Subclaim 2. Let $I_{i,j}$ be a bad interval corresponding to a vertex in $\mathcal{B}$. Then for every $i < t < j$, $g(t) - g(t-1) \leq g(t+1) - g(t)$.

Proof. By definition of $\mathcal{B}$, the parent of $I_{i,j}$ is good (the parent exists by our assumption that $I_{0,n} \notin \mathcal{B}$). Hence

$$f(i+1) - f(i) \leq \frac{f(j-1) - f(i+1)}{(j-1)-(i+1)} \leq f(j) - f(j-1). \tag{3}$$

By definition of $g(\cdot)$, $g(i+1) - g(i) = f(i+1) - f(i)$, $g(j) - g(j-1) = f(j) - f(j-1)$, and for every $i+1 < t \leq j-1$, $g(t) - g(t-1) = \frac{f(j-1)-f(i+1)}{(j-1)-(i+1)}$. Therefore, for every $i+1 < t < j-1$, $g(t) - g(t-1) = g(t+1) - g(t)$, and for both $t = i+1$ and $t = j-1$, we have $g(t) - g(t-1) \leq g(t+1) - g(t)$ as required.

Subclaim 3. The function g is convex.

Proof. We shall first show that all intervals $I_{i,j}$ corresponding to vertices in the tree are good with respect to g, and from this we derive the convexity of g.

We start with the first part. Consider any such interval $I_{i,j}$ whose corresponding vertex in the tree is v. Let $\mathit{Anchor} = \{i, i+1, \mathit{mid}-1, \mathit{mid}, \mathit{mid}+1, \mathit{mid}+2, j-1, j\}$ be the set of points which participate in the definition of a good interval $I_{i,j}$. We will show that the value of g on points $p \in \mathit{Anchor}$ is such that the interval $I_{i,j}$ is good with respect to g. There are two cases:

1. The interval $I_{i,j}$ is good with respect to f, and v does not have any ancestors in $\mathcal{B}$. If v also has no descendents in $\mathcal{B}$, then it clearly remains good with respect to g, since no modification is performed on any point in the interval, and so $g(t) = f(t)$ for every $i \leq t \leq j$. Otherwise, v has a descendent in $\mathcal{B}$. In this case, let $p \in \mathit{Anchor}$, let v' be a descendent of v, and let $I_{i',j'}$ denote

the interval corresponding to v'. If $i' \le p \le j'$, then by definition of the tree, either $p = i'$ or $p = i' + 1$ or $p = j' - 1$ or $p = j'$. Therefore, even if $v' \in \mathcal{B}$ and the interval $I_{i',j'}$ is modified, then by the definition of g we have that $g(p) = f(p)$ for every $p \in Anchor$. Thus $I_{i,j}$ remains good with respect to g.

2. Either $v \in \mathcal{B}$ or v has an ancestor in $\mathcal{B}$. In the former case, let $v' = v$, and in the latter case let v' be the ancestor that v has in $\mathcal{B}$. Let $I_{i',j'}$ be the corresponding interval of v'. By definition, $I_{i,j} \subseteq I_{i',j'}$. By Subclaim 2, $g(t) - g(t-1) \le g(t+1) - g(t)$ for every $i' < t < j'$, and in particular for every $i < t < j$. It follows that $I_{i,j}$ is good with respect to g.

Hence all intervals corresponding to vertices in the tree are good with respect to g. We now prove that for every $0 < t < n$ it holds that $g(t) - g(t-1) \le g(t+1) - g(t)$, and thus g is convex. Let $I_{i,j}$ be the smallest interval in the tree such that $i < t < j$. If $j - i \le 7$, then we are done, since the goodness of $I_{i,j}$ in this case means that g is convex over the whole interval. Otherwise, either $t = mid$ or $t = mid + 1$, where $mid = \lfloor (i+j)/2 \rfloor$. To verify this, note that if this were not the case, then either $i < t < mid$ or $mid + 1 < t < j$. Hence t is contained in a smaller interval in the tree, contradicting the minimality of $I_{i,j}$. But since $I_{i,j}$ is good with respect to g, $g(mid) - g(mid-1) \le g(mid+1) - g(mid)$, and $g(mid+1) - g(mid) \le g(mid+2) - g(mid+1)$. Thus we are done with the proof of Subclaim 3, and Theorem 1 follows. □

3. Building blocks for our algorithms for testing inverse monge. From this point on we focus on inverse Monge matrices. Analogous claims hold for Monge matrices. We also assume for simplicity that the dimensions of the matrices are $n_1 = n_2 = n$. In what follows we provide a characterization of inverse Monge matrices that is exploited by our algorithms. Given any real valued matrix $V = \{v_{i,j}\}_{i,j=0}^{i,j=n}$ we define an $(n+1) \times (n+1)$ matrix $C'_V = \{c_{i,j}\}_{i,j=0}^{i,j=n}$ as follows:

- $c_{0,0} = v_{0,0}$.
- For $i > 0$: $c_{i,0} = v_{i,0} - v_{i-1,0}$.
- For $j > 0$: $c_{0,j} = v_{0,j} - v_{0,j-1}$.
- And for every $i, j > 0$,

$$\begin{aligned} c_{i,j} &= (v_{i,j} - v_{i-1,j}) - (v_{i,j-1} - v_{i-1,j-1}) \\ &= (v_{i,j} - v_{i,j-1}) - (v_{i-1,j} - v_{i-1,j-1}). \end{aligned} \tag{4}$$

Let $C_V = \{c_{i,j}\}_{i,j=1}^{i,j=n}$ be the submatrix of C'_V that includes all but the first (0th) row and column of C'_V. The following two claims are well known and easy to verify. We include their proofs for completeness.

CLAIM 2. *For every* $0 \le i, j \le n$, $v_{i,j} = \sum_{k=0}^{i} \sum_{\ell=0}^{j} c_{k,\ell}$.

Proof. The claim is proved by induction on i and j.

The base case $i, j = 0$ holds by definition of $c_{0,0}$.

Consider any $i > 0$ and assume that the claim holds for every $k < i$, $j = 0$. We prove it for i and for $j = 0$. By definition of $c_{i,0}$ we have $v_{i,0} = v_{i-1,0} + c_{i,0}$. By the induction hypothesis, $v_{i-1,0} = \sum_{k=0}^{i-1} c_{k,0}$, and the induction step follows. The claim is similarly proved for every $j > 0$ and $i = 0$.

Finally, consider any $i, j > 0$ and assume that the claim holds for every $k < i$ and $\ell \le j$, and for every $k \le i$ and $\ell < j$. We prove it for i, j. By definition of $c_{i,j}$,

$v_{i,j} = v_{i-1,j} + (v_{i,j-1} - v_{i-1,j-1}) + c_{i,j}$. By the induction hypothesis,

$$v_{i-1,j} + (v_{i,j-1} - v_{i-1,j-1}) = \sum_{k=0}^{i-1}\sum_{\ell=0}^{j} c_{k,\ell} + \sum_{\ell=0}^{j-1} c_{i,\ell},$$

and the induction step follows. ∎

CLAIM 3. *A matrix V is an inverse Monge matrix if and only if C_V is a nonnegative matrix.*

Proof. If V is an inverse Monge matrix, then, in particular, for every $i, j \geq 1$ we have that $v_{i,j} + v_{i-1,j-1} \geq v_{i,j-1} + v_{i-1,j}$, which is equivalent to the condition $c_{i,j} \geq 0$.

In the other direction, consider any two points (i, j) and (i', j') such that $0 \leq i < i' \leq n$, $0 \leq j < j' \leq n$. Using Claim 2 we obtain

$$\begin{aligned} & v_{i',j'} - v_{i',j} - v_{i,j'} + v_{i,j} \\ &= \sum_{k=0}^{i'}\sum_{\ell=0}^{j'} c_{k,\ell} - \sum_{k=0}^{i'}\sum_{\ell=0}^{j} c_{k,\ell} - \sum_{k=0}^{i}\sum_{\ell=0}^{j'} c_{k,\ell} + \sum_{k=0}^{i}\sum_{\ell=0}^{j} c_{k,\ell} \\ &= \sum_{k=i+1}^{i'}\sum_{\ell=j+1}^{j'} c_{k,\ell}. \end{aligned} \tag{5}$$

But C_V is nonnegative, and therefore $v_{i',j'} - v_{i',j} - v_{i,j'} + v_{i,j} \geq 0$ as required. ∎

It follows from Claim 3 that if we find some entry of C_V that is negative, then we have evidence that V is not an inverse Monge matrix. However, it is not necessarily true that if V is far from being an inverse Monge matrix, then C_V contains many negative entries. For example, suppose that C_V is 1 in all entries except the entry $c_{n/2,n/2}$ which is $-n^2$. Then it can be verified that V is very far from being an inverse Monge matrix (this can be proved by showing that there are $\Theta(n^2)$ disjoint quadruples $v_{i,j}, v_{i',j'}, v_{i,j'}, v_{i',j}$ in V such that from any such quadruple at least one value should be changed in order to transform V into an inverse Monge matrix). However, as our analysis will show, in such a case there are many submatrices in C_V whose sum of elements is negative. Thus our testing algorithms will sample certain submatrices of C_V and check that the sum of elements in each submatrix sampled is nonnegative. We first observe that it is possible to check this efficiently.

CLAIM 4. *Given access to V it is possible to check in time $O(1)$ if the sum of elements in a given submatrix A of C_V is nonnegative. In particular, if the lower-left entry of A is (i, j) and its upper-right entry is (i', j'), then the sum of elements of A is $v_{i',j'} - v_{i',j-1} - v_{i-1,j'} + v_{i-1,j-1}$.*

Proof. Assume that $A = (c_{k,\ell})_{k=i,\ell=j}^{k=i',\ell=j'}$ is a submatrix of C_V. Recall that for any q, p, we have $v_{q,p} = \sum_{k=0}^{q}\sum_{\ell=0}^{p} c_{k,\ell}$. Thus the sum of elements of A is

$$\begin{aligned} \sum_{k=i}^{i'}\sum_{\ell=j}^{j'} c_{k,\ell} &= \sum_{k=0}^{i'}\sum_{\ell=j}^{j'} c_{k,\ell} - \sum_{k=0}^{i-1}\sum_{\ell=j}^{j'} c_{k,\ell} \\ &= \left(\sum_{k=0}^{i'}\sum_{\ell=0}^{j'} c_{k,\ell} - \sum_{k=0}^{i'}\sum_{\ell=0}^{j-1} c_{k,\ell}\right) - \left(\sum_{k=0}^{i-1}\sum_{\ell=0}^{j'} c_{k,\ell} - \sum_{k=0}^{i-1}\sum_{\ell=0}^{j-1} c_{k,\ell}\right) \\ &= (v_{i',j'} - v_{i',j-1}) - (v_{i-1,j'} - v_{i-1,j-1}) \,. \end{aligned}$$

Therefore computing the sum of elements of any submatrix A of C_V can be done by checking only four entries in the matrix V. ∎

3.1. Filling submatrices. An important building block for the analysis of our algorithms is a procedure for "filling in" a submatrix. That is, given constraints on the sum of elements in each row and column of a given submatrix, we are interested in assigning values to the entries of the submatrix so that these constraints are met.

Specifically, let $a_1, \ldots, a_s$ and $b_1, \ldots, b_t$ be nonnegative real numbers such that $\sum_{i=1}^{s} a_i \geq \sum_{j=1}^{t} b_j$. Then it is possible to construct an $s \times t$ nonnegative real matrix T such that the sum of elements in column j is exactly b_j and the sum of elements in row i is at most a_i. In the special case that $\sum_{i=1}^{s} a_i = \sum_{j=1}^{t} b_j$, the sum of elements in row i will equal a_i. In particular, this can be done by applying the following procedure, which is the same as the one applied to obtain an initial feasible solution for the linear-programming formulation of the transportation problem.

PROCEDURE 2 (fill matrix $T = (t_{i,j})_{i,j=1}^{i=s,j=t}$).
Initialize $\bar{a}_i = a_i$ *for* $i = 1, \ldots, s$ *and* $\bar{b}_j = b_j$ *for* $j = 1, \ldots, t$.
(In each of the following iterations, $\bar{a}_i$ *is an upper bound on what remains to be filled in row* i*, and* $\bar{b}_j$ *is what remains to be filled in column* j*.)*
For $j = 1,\ldots,t$*:*
  For $i = 1,\ldots,s$*:*
    Assign to entry (i, j) *the value* $x = \min\{\bar{a}_i, \bar{b}_j\}$.
    Update $\bar{a}_i = \bar{a}_i - x$, $\bar{b}_j = \bar{b}_j - x$.

CLAIM 5. *Procedure 2 fills the matrix* T *with nonnegative values* $t_{i,j}$ *such that at the end of the procedure,* $\sum_{i=1}^{s} t_{i,j} = b_j$ *for every* $j = 1, \ldots, t$*, and* $\sum_{j=1}^{t} t_{i,j} \leq a_i$ *for every* $i = 1, \ldots, s$*. If initially* $\sum_{j=1}^{t} b_j = \sum_{i=1}^{s} a_i$*, then* $\sum_{j=1}^{t} t_{i,j} = a_i$ *for every* $i = 1, \ldots, s$.

Proof. Notice that initially $\bar{a}_i = a_i \geq 0$ and $\bar{b}_j = b_j \geq 0$. Thus when we update $\bar{a}_i = \bar{a}_i - x = \bar{a}_i - \min\{\bar{a}_i, \bar{b}_j\} \geq 0$ and similarly $\bar{b}_j = \bar{b}_j - x = \bar{b}_j - \min\{\bar{a}_i, \bar{b}_j\} \geq 0$. Therefore the $\bar{a}_i$'s and $\bar{b}_j$'s are always nonnegative. Hence all values x filled in T are nonnegative, since $x = \min\{\bar{a}_i, \bar{b}_j\} \geq 0$. Furthermore, after each such update the new sum over the $\bar{a}_i$'s equals the old sum over the $\bar{a}_i$'s minus x, and a similar statement holds for the sum over the $\bar{b}_j$'s. Thus at all stages of the procedure, $\sum_{i=1}^{s} \bar{a}_i \geq \sum_{j=1}^{t} \bar{b}_j$, and if initially $\sum_{i=1}^{s} a_i = \sum_{j=1}^{t} b_j$, then $\sum_{i=1}^{s} \bar{a}_i = \sum_{j=1}^{t} \bar{b}_j$.

We now show that the sum of elements in each column is as required. Observe that the procedure fills the columns one by one. Therefore when we start to fill column j we have $\bar{b}_j = b_j$. Since $\sum_{i=1}^{s} \bar{a}_i \geq \sum_{j=1}^{t} \bar{b}_j$ at this stage, and all $\bar{a}_i$'s are nonnegative, necessarily, $\sum_{i=1}^{s} \bar{a}_i \geq \bar{b}_j = b_j$. Let $1 \leq k \leq s$ be the minimum integer such that $\sum_{i=1}^{k} \bar{a}_i \geq b_j$. Then by definition of the procedure, for every $i < k$, the entry (i, j) is filled with the value $\bar{a}_i$, and the entry $(k + 1, j)$ is filled with the value $b_j - \sum_{i=1}^{k} \bar{a}_i$. The total is hence b_j as required.

As for the rows, at all stages $\bar{a}_i$ equals a_i minus the sum of all elements filled so far in row i. Therefore since $\bar{a}_i \geq 0$, then the sum of elements in row i is at most a_i. Furthermore, if initially $\sum_{i=1}^{s} a_i = \sum_{j=1}^{t} b_j$, then the sum of elements in row i will be exactly a_i. To show this note that at the end of the procedure, $\sum_{j=1}^{t} \bar{b}_j = 0$, since each $\bar{b}_j$ equals b_j minus the sum of all elements in column j, and we have shown that the sum of elements in column j is b_j. But $\sum_{i=1}^{s} \bar{a}_i = \sum_{j=1}^{t} \bar{b}_j$, and therefore also $\sum_{i=1}^{s} \bar{a}_i = 0$ at the end. Since $\bar{a}_i \geq 0$, this means that $\bar{a}_i = 0$. Hence the sum of elements in row i must be a_i. ∎

4. A testing algorithm for inverse monge matrices. We first present a simple algorithm for testing if a matrix V is an inverse Monge matrix whose running

time is $O(n/\epsilon)$. In the next section we show a significantly faster algorithm that is partly based on the ideas presented here. We may assume without loss of generality that n is a power of 2. This is true since our algorithms probe the coefficients matrix C_V, and we may simply "pad" it by 0's to obtain rows and columns that have lengths which are powers of 2 and run the algorithm with $\epsilon \leftarrow \epsilon/4$. We shall need the following two definitions for both algorithms.

DEFINITION 4 (subrows, subcolumns, and submatrices). *A* subrow *in an $n \times n$ matrix is a consecutive sequence of entries that belong to the same row. The subrow $((i,j),(i,j+1),\ldots,(i,j+t-1))$ is denoted by $[\,]_{i,j}^{1,t}$. A* subcolumn *is defined analogously and is denoted by $[\,]_{i,j}^{s,1} = ((i,j),(i+1,j),\ldots,(i+s-1,j))$. More generally, an $s \times t$ submatrix whose bottom-left entry is (i,j) is denoted $[\,]_{i,j}^{s,t}$.*

DEFINITION 5 (legal submatrices). *A subrow in an $n \times n$ matrix is a* legal *subrow if it can result from bisecting the row of length n that contains it in a recursive manner. That is, a complete (length n) row is legal, and if $[\,]_{i,j}^{1,t}$ is legal, then so are $[\,]_{i,j}^{1,t/2}$ and $[\,]_{i,j+t/2}^{1,t/2}$. A legal subcolumn is defined analogously. A submatrix is* legal *if both its rows and its columns are legal.*

Note that the legality of a subrow $[\,]_{i,j}^{1,t}$ is not dependent on the actual row i it belongs to, but rather it depends on its starting position j and ending position $j+t-1$ within its row. An analogous statement holds for legal subcolumns. See also Figure 1 for an illustration of the concept of legal submatrices.

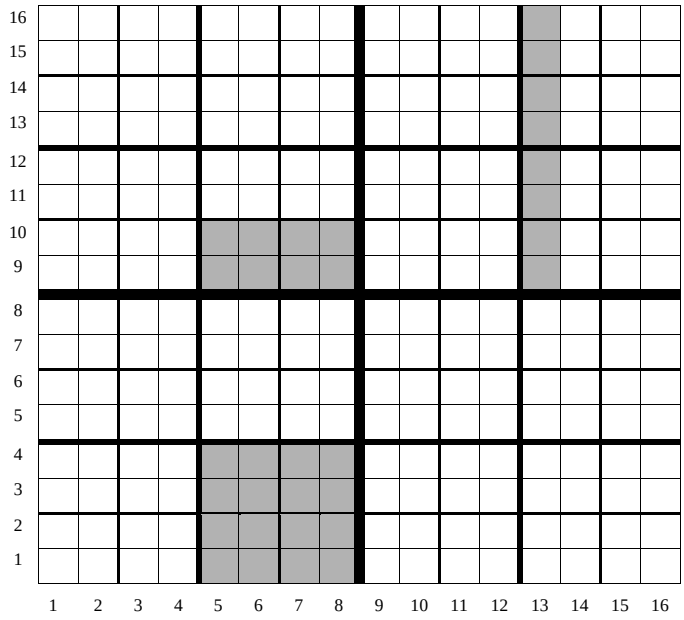


FIG. 1. *An illustration of three legal submatrices. One of the legal submatrices is a square (4×4) submatrix, and the other two are rectangular (but legal) submatrices. The 8×1 submatrix on the top-right is a legal subcolumn.*

Although a submatrix is just a collection of positions (entries) in an $n \times n$ matrix, we talk throughout the paper about sums of elements in certain submatrices A of C_V. In this we mean the sum of elements of C_V determined by the set of positions in A.

DEFINITION 6 (good and bad submatrices). *We say that a submatrix A of C_V is*

good *if the sum of elements in each row of A is nonnegative and the sum of elements in each column of A is nonnegative. Otherwise, A is* bad.

DEFINITION 7 (good and bad points). *We say that point (i, j) is* good *if all legal* square *submatrices A of C_V which contain (i, j) are good. Otherwise, the point is* bad.

ALGORITHM 2 (Test-Monge I).

1. *Choose $8/\epsilon$ points in the matrix C_V and check that they are good.*
2. *If all points are good, then accept; otherwise, reject.*

By Claim 4, it is possible to check in constant time that the sum of elements in a subrow (subcolumn) of C_V is nonnegative. Therefore, it is possible to test that an $s \times s$ square submatrix A of C_V is good in time $\Theta(s)$. Notice that every point in an $n \times n$ matrix is contained in $\log n$ legal square submatrices. Hence the time required to check whether a point is good is $O(n)+O(n/2)+\cdots+O(n/2^i)+\cdots+O(1) = O(n)$, and the complexity of the algorithm is $O(n/\epsilon)$.

THEOREM 2. *If V is an inverse Monge matrix, then Algorithm* 2 *always accepts, and if V is ϵ-far from being an inverse Monge matrix, then Algorithm* 2 *rejects with probability at least* $2/3$.

Proof. The first part of the theorem follows directly from Claim 3. In order to prove the second part of the theorem, we show that if V is ϵ-far from being inverse Monge, then C_V contains more than $(\epsilon/4)n^2$ bad points. The second part of the theorem directly follows because the probability in such a case that no bad point is selected by the algorithm is at most $(1-\epsilon/4)^{(8/\epsilon)} < e^{-2} < 1/3$.

Assume contrary to the claim that C_V contains at most $(\epsilon/4)n^2$ bad points. We shall show that by modifying at most ϵn^2 entries in V we obtain an inverse Monge matrix (in contradiction to our assumption concerning V). Let us look at the set of bad points in C_V, and for each such bad point look at the largest bad legal square submatrix in C_V that contains this bad point. By our assumption on the number of bad points, it must be the case that the area of all these maximal bad submatrices is at most $(\epsilon/4)n^2$, because all the points in a bad submatrix are bad.

For each maximal bad legal square submatrix B of C_V we will look at the legal square submatrix A that contains B. By definition of legal square submatrices, the matrix A is uniquely defined. By the maximality of B, the submatrix A must be good. Indeed, since B is maximal, if it is of size $s \times s$, where $s < n$, then the legal square submatrix of size $2s \times 2s$ that contains it must be good. But if $s = n$, then $B = C_V$, implying that all n^2 points in C_V are bad, contradicting our assumption on the number of bad points.

Next observe that every two different maximal bad legal square submatrices B and B' are disjoint. This is true since every two different legal square submatrices are either disjoint or one is contained in the other. Combining this with the fact that for each maximal bad legal square submatrix we take the good square legal submatrix that is four times its size, the area of the union of all these good submatrices is at most $4 \cdot (\epsilon/4)n^2 = \epsilon n^2$.

Turning to the collection of resulting good submatrices, note that every two of these submatrices are either disjoint, or are exactly the same, or one is contained in the other. If a good submatrix is strictly contained in another one, then we ignore it and deal only with the larger good submatrix containing it. Thus we have a set of disjoint good submatrices that contain all negative entries in the matrix. For each of these good submatrices A, we modify A so that it contains only nonnegative elements, and the sum of elements in each row and column of A remains as it was. This can

be done by applying Procedure 2 to A as described in section 3.1 (using the actual (nonnegative) sums of rows and columns of A as the input to the procedure).

Note that after modifying all these good submatrices of C_V, the new matrix C_V is nonnegative, and thus the corresponding new matrix V must be an inverse Monge matrix. It remains to show that at most ϵn^2 values were changed in V following the changes to C_V. Notice that we made sure that the sum of elements in each row and column of each modified submatrix A remains as it was. Therefore the values of all points $v_{k,\ell}$ in V that are outside A are not affected by the change to A, since by Claim 2 we have that $v_{k,\ell} = \sum_{i=0}^{k} \sum_{j=0}^{\ell} c_{i,j}$. □

5. A faster algorithm for inverse monge matrices. Algorithm 2 described above has running time linear in n, which is already sublinear in the size of the matrix, n^2. In this section we show how to significantly improve the dependence on n. We present a variant of the algorithm whose running time is $O(\epsilon^{-1} \log^2 n)$. The new algorithm will be based on a similar principle as that of Algorithm 2. That is, it will uniformly select points and verify that certain submatrices that contain them are good. However, there will be two main differences which we now describe briefly.

Algorithm 2 suffers from a relatively slow running time, since for each submatrix that the algorithm checks, it verifies that the sum of elements in *every* row and column is nonnegative. Therefore, we first relax the concept of a good submatrix and demand only that the sum of *all* its elements be nonnegative (instead of the sum of every row and column). This change, however, requires us to check for each point selected by the algorithm, not only that the legal *square* submatrices which contain it are good, but rather to verify that *all* legal submatrices that contain the point are good. Actually, we check something slightly stronger: The algorithm will verify for each legal submatrix T that it examines that the four legal equal-size submatrices that reside within T and are half of T's length in each dimension are good as well. In order to formalize the above, we first redefine the concepts of good (bad) submatrices and good (bad) points, and introduce the notion of tainted submatrices and tainted points.

DEFINITION 8 (good and bad submatrices and points). *A (legal) submatrix T of C_V is* good *if the sum of all its elements is nonnegative. Otherwise, T is* bad.

A point is good *if every legal submatrix of C_V that contains it is good. Otherwise, the point is* bad.

DEFINITION 9 (tainted submatrices and points). *A good legal submatrix T of C_V is* tainted *if any one of the four legal submatrices that it contains and that are half its height and half its width is bad. A point is* tainted *if some legal submatrix that contains it is tainted.*

Note that every bad point is tainted, but good points may be tainted as well.

For the sake of the presentation, we shall assume that every row and every column in C_V (that is, every subrow and subcolumn of length n) have nonnegative sums. In subsection 5.2 we explain how to remove this assumption. Note that this assumption implies that every $s \times n$ submatrix is good, and similarly every $n \times s$ submatrix is good (but of course it has no implications on smaller submatrices).

ALGORITHM 3 (Test-Monge II).

1. *Uniformly select $2/\epsilon$ points in the matrix C_V and check for each of them whether it is tainted.*
2. *If no point selected is tainted, then accept; otherwise, reject.*

Note that by Definition 5, each point in an $n \times n$ matrix is contained in $O(\log^2 n)$ legal submatrices. Thus by Claim 4, checking whether a point is tainted takes time

$O(\log^2 n)$. Therefore the running time of the algorithm is $O((\log^2 n)/\epsilon)$.

THEOREM 3. *If V is an inverse Monge matrix, then Algorithm* 3 *always accepts, and if V is ϵ-far from being an inverse Monge matrix, then Algorithm* 3 *rejects with probability at least* 2/3.

5.1. Outline of the proof of Theorem 3. If V is an inverse Monge matrix, then by Claim 3 all elements in C_V are nonnegative. This directly implies that all (legal) submatrices are good, and so all points are good and are not tainted. Hence in this case the algorithm always accepts. Suppose that V is ϵ-far from being inverse Monge. We claim that in such a case C_V must contain more than ϵn^2 tainted points, causing the algorithm to reject with probability at least

$$1 - (1-\epsilon)^{(2/\epsilon)} > 1 - e^{-2} > 2/3.$$

Assume contrary to the claim that C_V contains at most ϵn^2 tainted points. Our goal from this point on is to show that in such a case V is ϵ-close to being an inverse Monge matrix.

The proof of this part will follow along similar lines to those used in the proof of Theorem 2. That is, we consider all maximal bad legal submatrices of C_V, and for each such bad submatrix we consider the legal good submatrix that is four times its area and contains it. Once again, this submatrix is unique. By Definition 9, this submatrix is tainted. We then take the union of all these good but tainted submatrices. By our assumption on the number of tainted points, the area of this union is at most ϵn^2 since all points in the union are tainted.

Finally, we show how to modify the values in this union so that the resulting matrix is an inverse Monge matrix. This time, however, since the maximal bad submatrices may intersect (which was not the case in the slower algorithm), the good tainted submatrices that contain them may intersect in nontrivial ways (that is, not only by coinciding or by strict containment). As a result, the union of the good submatrices has a possibly complex structure (and in particular it is no longer a simple union of disjoint submatrices), and the process of properly modifying this union is much more involved. We now describe precisely the necessary definitions and proceed with a detailed proof.

DEFINITION 10 (maximal bad legal submatrix). *A bad legal submatrix T of C_V is a* maximal bad legal submatrix *of C_V if it is not contained in any larger bad legal submatrix of C_V.*

Now consider all maximal bad legal submatrices of C_V. Note that every negative entry in C_V is contained in the union of these bad submatrices. For each such submatrix B let us take the (unique) legal submatrix T that contains it and has twice the number of rows and twice the number of columns of B (by our assumption that all full rows and columns have a nonnegative sum it is indeed possible to double the rows and columns of B). Then by the maximality of B, the resulting submatrix is good. We now take the union of all these good (but tainted) legal submatrices. Recall that the area of the union of all tainted (legal) submatrices of C_V is at most ϵn^2. Denote the union of all these good tainted submatrices by R. See, for example, Figure 2.

In subsections 5.3 and 5.4 we show that it is possible to change the (at most ϵn^2) entries of C_V within R to nonnegative values so that the following property holds.

PROPERTY 1 (sum property for R). *For every point (i,j) outside of R, the sum of the elements in the modified entries (i',j') within R such that $i' \le i$ and $j' \le j$ is the same as in the original matrix C_V.*

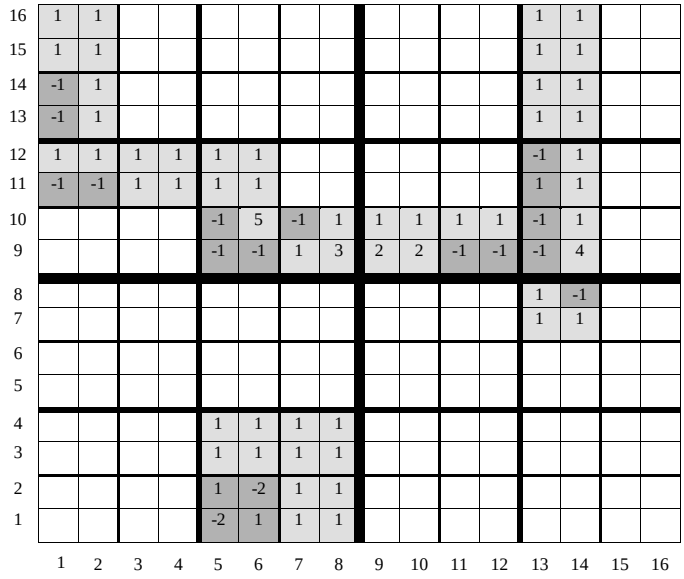


FIG. 2. *An example of the structure of a subset R, where R is the union of all gray cells in the matrix (both dark and light gray). All values in cells outside of R are nonnegative and are not displayed for the sake of simplicity. The bad legal submatrices determining R are the dark gray submatrices. Each is contained inside a good but tainted legal submatrix that has twice the number of rows and twice the number of columns (good tainted submatrices are marked both by light and dark gray). For example, there is a bad submatrix in column* 1*, rows* 13 *and* 14*, and the good legal submatrix containing it is the submatrix over columns* 1 *and* 2 *and rows* 13 *through* 16*. Observe that maximal bad legal submatrices may intersect. For example, the bad submatrix containing the two cells in row* 9 *and columns* 5 *and* 6 *intersects with the bad submatrix containing the two cells in column* 5 *and rows* 9 *and* 10*. Their corresponding good submatrices also intersect.*

Let $\tilde{C}_V$ be the matrix obtained from C_V by modifying R so that Property 1 holds, and let $\tilde{V}$ be the matrix which corresponds to $\tilde{C}_V$. Then it follows from Claim 2 that $\tilde{V}$ is at most ϵ-far from the original matrix V, and this completes the proof of Theorem 3. Before we continue with showing how to obtain Property 1, we explain shortly how to remove the assumption that all (full) rows and columns in C_V have a nonnegative sum.

5.2. Dealing with rows/columns having a negative sum. Suppose first that $\epsilon \leq 4/n$. Then we may directly check in time $O(1/\epsilon)$ that in fact all rows and columns of the matrix C_V have nonnegative sums (using Claim 4) and reject if some row or column has a negative sum. Hence in this case our assumption is valid. Thus assume that $\epsilon > 4/n$.

First we slightly modify Algorithm 3 so that it uniformly selects $4/\epsilon$ points in C_V (instead of $2/\epsilon$). In such a case, if C_V contains more than $(\epsilon/2)n^2$ tainted points, then the algorithm rejects with probability at least $2/3$. We thus assume that C_V contains at most $(\epsilon/2)n^2$ tainted points and strive to show that in such a case V is ϵ-close to being an inverse Monge matrix. Since we do not assume that every row and column in C_V has a nonnegative sum, we first modify C_V so that it has this property.

Consider each row i in C_V whose sum of elements in negative. Suppose that we modify the last entry in the row, $c_{i,n}$, so that the new sum of all elements is 0. Similarly, we modify the last entry $c_{n,j}$ in each column j that has a negative sum. Let $\bar{C}_V$ be the resulting matrix, and let $\bar{V}$ be the matrix corresponding to $\bar{C}_V$. Then

all rows and columns in $\bar{C}_V$ have a nonnegative sum, and by Claim 2 $\bar{V}$ and V differ on at most $2n-1 < (\epsilon/2)n^2$ entries (at most all elements in the last column and last row).

Now we may define the region R as we did in the previous subsection. Note that in this case the area of the region R is at most $(\epsilon/2)n^2$. We can therefore continue in proving that it is possible to modify only the elements within R so that they are all nonnegative and Property 1 holds. This will imply that the total number of entries that should be modified (first to obtain nonnegative rows and columns, and then to refill R) is at most ϵn^2, as desired.

5.3. Refilling R to obtain Property 1. Let R be as defined in section 5.1. Recall that R consists of a union of good legal submatrices. (The fact that they are tainted is no longer relevant.) In the following discussion, when we talk about elements in submatrices of R we mean the elements in C_V determined by the corresponding set of positions in R.

We are interested in refilling the entries in R with *nonnegative* values so that Property 1 will hold. Note that if R is just a submatrix (block) of C_V, then we can use Procedure 2 to refill R as desired. However, in general the structure of R is more complex. We show that there is a way to partition R into disjoint blocks and refill each block using Procedure 2. In subsection 5.3.1 we define precisely what blocks are and present several other notions that are needed for the refilling procedure. The refilling procedure for R is described in subsection 5.3.2, and its correctness is proved in subsection 5.4.

5.3.1. Preliminaries for the refilling procedure. As stated above, the refilling procedure will partition R into disjoint blocks (submatrices) and fill each block separately with nonnegative values so that Property 1 is maintained. We start with defining the following term that will be needed to define blocks.

DEFINITION 11 (maximal (legal) subrow/column). *Given a subset R of entries in an $n \times n$ matrix, a subrow T is a* maximal (legal) subrow with respect to R *if T is contained in R and there is no larger (legal) subrow T' such that $T \subset T' \subseteq R$. A* maximal (legal) subcolumn with respect to R *is defined analogously.*

For the sake of succinctness, whenever it is clear what R is, we shall just say maximal (legal) subrow and drop the suffix "with respect to R." Note that a maximal subrow is simply a maximal consecutive sequence of entries in R that belong to the same row, while a maximal legal subrow is a more constrained notion. In particular, a maximal subrow may be a concatenation of several maximal legal subrows. We can now define blocks as follows.

DEFINITION 12 (maximal block). *A maximal block $B = [\,]_{i,j}^{s,t}$ in R is a submatrix contained in R which has the following property: It consists of a maximal consecutive sequence of maximal legal subcolumns of the same height. The maximality of each subcolumn is as in Definition* 11. *That is, for every $j \le r \le j+t-1$, the column $[\,]_{i,r}^{s,1}$ is a maximal legal subcolumn (with respect to R).*

The height *of a maximal block B is the height of the columns in B (equivalently, the number of rows in B).*

The maximality of the sequence of subcolumns in a block $B = [\,]_{i,j}^{s,t}$ means that we can extend the sequence of columns neither to the left nor to the right. That is, neither $[\,]_{i,j-1}^{s,1}$ nor $[\,]_{i,j+t}^{s,1}$ is a maximal legal subcolumn in R. (Specifically, each either is not fully contained in R or R contains a larger legal subcolumn that contains it.)

We shall sometimes refer to maximal blocks simply as blocks. Observe that by this definition, R is indeed partitioned in a unique way into maximal disjoint blocks.

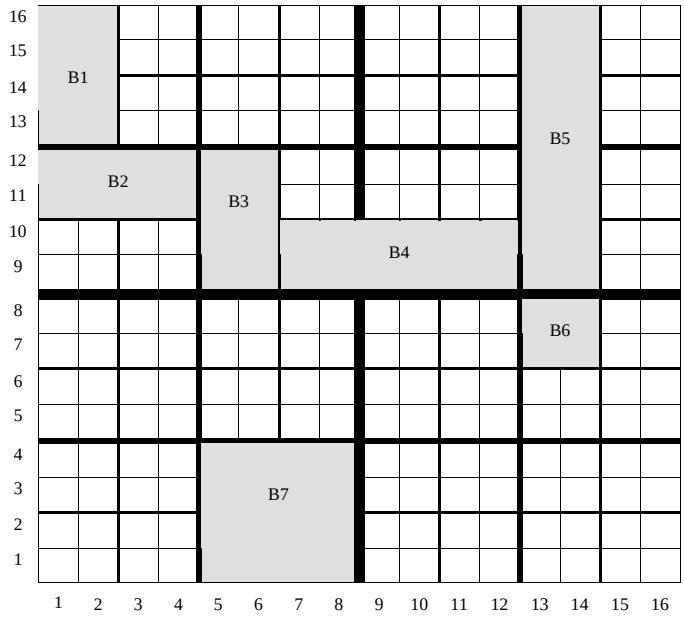


FIG. 3. *An example of the partition of R shown in Figure* 2 *into maximal blocks (numbered B_1–B_7). Note that the ratio between the heights of any two blocks is always a power of* 2. *Furthermore, the blocks are aligned in the following way. Suppose a block B has height s, and a block B' has height $s' \leq s$ and some of their subrows belong to the same row of the matrix (e.g., B_3 and B_4, or B_4 and B_5). Then the shorter block B' must be aligned with either the first or second half of B, or with one of the quarters of B, or with one of its eighths, etc.*

See Figure 3 for an illustration to how the subset R from Figure 2 is partitioned into maximal blocks.

Three additional notions that will be needed for the refilling procedure are defined below. The first two are illustrated in Figure 4.

DEFINITION 13 (covers). *We say that a submatrix A* covers *a given block B with respect to R if $B \subseteq A \subseteq R$ and the number of rows in A equals the height of B.*

We say that A is a maximal row-cover *with respect to R if A consists of maximal subrows with respect to R.*

DEFINITION 14 (borders). *We say that a submatrix $T = [\ ]_{i,j}^{s,t}$* borders *another submatrix $T' = [\ ]_{i',j'}^{s',t'}$ if $i' \leq i + s - 1$ and $i \leq i' + s' - 1$, and either $j' = j + t$ (so that T is to the left of T') or $j' + t' = j$ (so that T is to the right of T').*

DEFINITION 15 (sums). *For a given submatrix T, we denote the sum of the elements in T by $sum(T)$.*

5.3.2. The procedure for refilling *R*. We now describe the procedure that refills the entries of R with nonnegative values so as to obtain Property 1. Recall that R is a disjoint union of maximal blocks. Hence if we remove a maximal block from R, then the maximal blocks of the remaining structure are simply the remaining maximal blocks of R. For simplicity of this introductory discussion, after removing a block from R, we refer to the remaining structure as R. The procedure described below will remove the blocks of R one by one, in order of increasing (nondecreasing) height, and refill each block separately using Procedure 2.

Recall that when (re)filling an $s \times t$ submatrix, Procedure 2 is provided with non-

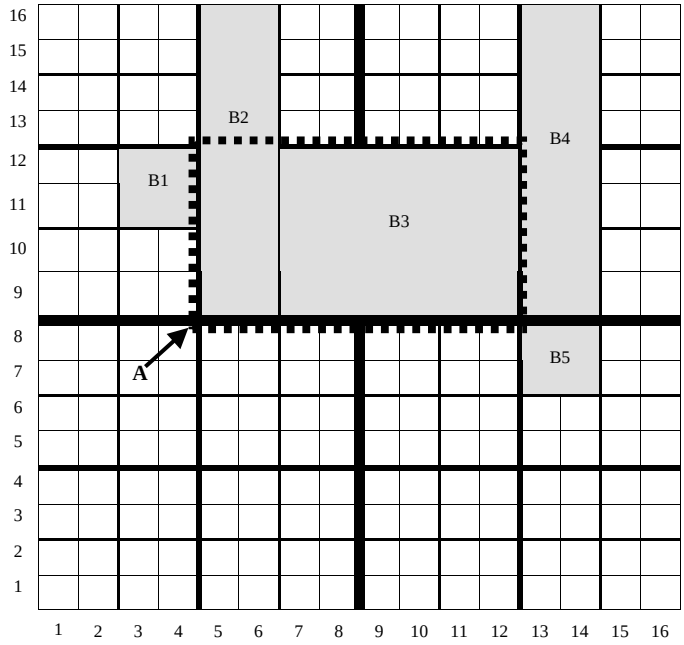


FIG. 4. *An illustration of the notions of covers and borders. Here the submatrix A (extending from row* 9 *to* 12 *and from column* 5 *to* 12*) covers the block* B_3 *(but is not a maximal row-cover with respect to R). The submatrix A borders block* B_1 *(from the left of A) and block* B_4 *(from the right of A).*

negative values $a_1, \ldots, a_s$ and $b_1, \ldots, b_t$ such that $\sum_{i=1}^{s} a_i \geq \sum_{j=1}^{t} b_j$. It then fills the submatrix with nonnegative values so that the sum of elements in column j is exactly b_j and the sum of elements in row i is at most a_i. Whenever we apply Procedure 2 to a block B, the column sums $b_1, \ldots, b_t$ are simply set to be the sums of the elements in the corresponding subcolumns of B in C_V. By definition of (maximal) blocks, these subcolumns are maximal legal subcolumns, and as we show in subsection 5.4.1, this ensures that their sums are nonnegative.

The setting of the upper bounds $a_1, \ldots, a_s$ for the row sums is a little more involved. At any point in the algorithm, each maximal subrow L is associated with a *designated* sum, denoted $\overline{sum}(L)$. This is the sum we intend it to have when the refilling procedure terminates. Initially, for every maximal subrow L in R, we set $\overline{sum}(L) = sum(L)$. That is, $\overline{sum}(L)$ is equal to the original sum of subrow L in C_V. In subsection 5.4.1 we show that these sums are all nonnegative. When refilling a block B, we first find the row-cover A of B that is a maximal row-cover with respect to (the current) R. Since the blocks are filled by order of height and blocks are removed after they are filled, such a maximal row-cover must exist when B is covered and is unique. We then use the designated sums of the (maximal) rows of A as the upper bounds $a_1, \ldots, a_s$ for the sums of rows of B. As we prove subsequently, it always holds that $\sum_{i=1}^{s} a_i \geq \sum_{j=1}^{t} b_j$ as required by Procedure 2. After removing a block B from R, we obtain new, shorter, maximal subrows in the remaining structure $R \setminus B$, and we must associate with these shorter subrows new designated sums. Procedure 2 is used here as well to determine how to set these designated row sums, in a manner explained in detail in step 3 below. For an illustration, see Figure 5.

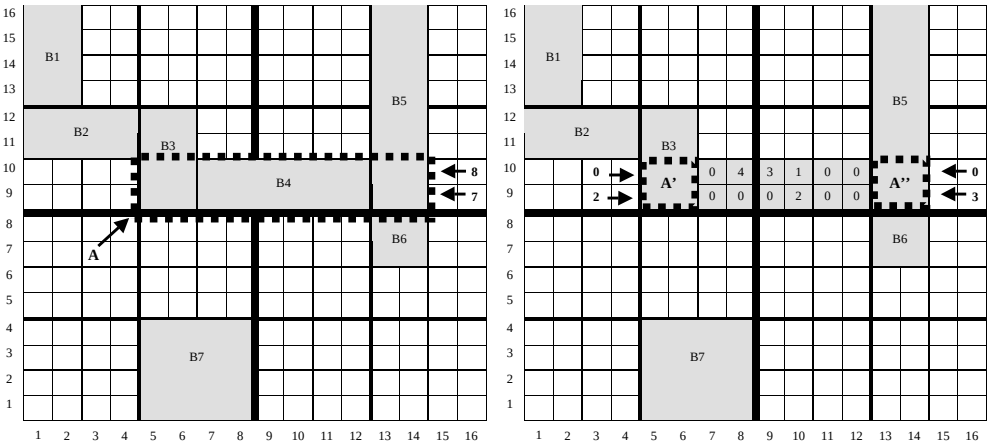


FIG. 5. *An illustration of one iteration of step* 3 *in Procedure* 3*, where we apply the procedure to the matrix illustrated in Figures* 2 *and* 3*. The first block filled may be either* B_2*,* B_4*, or* B_6 *(all three have height* 2*, which is the minimum among all blocks). Here we have selected to refill* B_4 *first. On the left we see the maximal row-cover* A *that covers* B_4*, where the designated sums of the two rows of* A *are* 8 *and* 7 *(in accordance with the values appearing in Figure* 2*). On the right we see the values that the Procedure* 2 *has entered in the cells of* B_4*. We also see the two submatrices,* A' *and* A''*, that remain of* A *after* B_4 *is removed from* R *and the designated sums of the new maximal rows in* A' *and* A''*.*

PROCEDURE 3 (refill R).

1. *We assign each maximal subrow* L *in* R *a designated sum of elements for that row, which is denoted by* $\overline{sum}(L)$*. Initially, we set* $\overline{sum}(L)$ *to be* $sum(L)$*.*
2. *Let* m *be the number of maximal blocks in* R*, and let* $R_1 = R$*.*
3. *For* $p = 1, \ldots, m$ *we do the following:*
   (a) *Let* B_p *be a maximal block in* R_p *whose height is minimum among all maximal blocks of* R_p*, and assume that* B_p *is an* $s \times t$ *submatrix. Let* A_p *be a maximal row-cover of* B_p *with respect to* R_p*. For* $1 \le \ell \le s$*, let* L_ℓ *denote the subrow of* A_p *that covers the* ℓ*th subrow of* B_p*.*
   (b) *Refill* B_p *by applying Procedure* 2 *(see section* 3.1*), where the sum filled in the* k*th subcolumn of* B_p*,* $1 \le k \le t$*, should be the original sum of this subcolumn in* C_V*, and the sum filled in the* ℓ*th subrow of* B_p*,* $1 \le \ell \le s$*, is at most* $\overline{sum}(L_\ell)$*.*
   For each $1 \le \ell \le s$*, let* x_ℓ *denote the sum of elements filled by Procedure* 2 *in the* ℓ*th subrow of* B_p*.*
   (c) *Let* $R_{p+1} = R_p \setminus B_p$*. We next assign designated sums to the rows of* R_{p+1} *that have been either shortened or broken into two parts by the removal of* B_p *from* R_p*. This is done as follows:*
   The set $A_p \setminus B_p$ *is the union of two nonconsecutive submatrices,* A' *and* A''*, so that* A' *borders* B_p *from the left of* B_p *and* A'' *borders* B_p *from the right of* B_p *(where it is possible that one or both of these submatrices does not exist). Let* L'_ℓ *and* L''_ℓ *be the subrows in* A' *and* A''*, respectively, that are contained in subrow* L_ℓ *of* A_p*. We assign to* L'_ℓ *and* L''_ℓ *nonnegative designated sums,* $\overline{sum}(L'_\ell)$ *and* $\overline{sum}(L''_\ell)$*, that satisfy the following:*

$$\overline{sum}(L'_\ell) + \overline{sum}(L''_\ell) = \overline{sum}(L_\ell) - x_\ell$$

and, furthermore,

$$\sum_{row\ L \in A'} \overline{sum}(L) = sum(A'), \qquad \sum_{row\ L \in A''} \overline{sum}(L) = sum(A'').$$

This is done by applying Procedure 2 *to a* $2 \times s$ *matrix whose sums of columns are* $sum(A')$ *and* $sum(A'')$ *and sums of rows are* $\overline{sum}(L_\ell) - x_\ell$, *where* $1 \le \ell \le s$.
(Note that one or both of A' *and* A'' *may not exist. This can happen if* B_p *bordered* $A_p \setminus B_p$ *on one side and its boundary coincided with* R_p *or if* $A_p = B_p$. *In this case, if, for example,* A' *does not exist, then we view it as a submatrix of height* 0, *where* $sum(A') = 0$.*)*

5.4. Proving that Procedure 3 is correct. In order to prove that Procedure 3 is correct we have to prove two claims. First, we have to show that the procedure does not "get stuck," namely, that all iterations of the procedure can be completed. Second, we have to prove that at the end of the procedure, the refilled structure R has Property 1. Before we prove these two claims we first prove some properties relating to the sum of elements in maximal blocks and other submatrices of R. These properties will be used to show that the procedure does not get stuck.

5.4.1. Sums of blocks and other submatrices. We first prove the following simple lemma regarding the sum of elements in maximal legal subrows and subcolumns of R.

LEMMA 6. *The sum of elements in every maximal legal subrow and every maximal legal subcolumn in* R *is nonnegative.*

Proof. We prove the lemma for maximal legal subrows. The claim for maximal legal subcolumns is analogous. Assume, contrary to the claim, that R contains some maximal legal subrow $L = [\ ]_{i,j}^{1,t}$ whose sum of elements is negative. Let T be the maximal bad legal submatrix in C_V that contains L. By the maximality of L, necessarily $T = [\ ]_{i',j}^{s,t}$ for some $i' \le i$ and $s \ge 1$. That is, the rows of T (one of which is L) are of length t. By the construction of R, R must contain a good legal submatrix T' that contains T and is twice as large in each dimension. But this contradicts the maximality of L. □

It directly follows from Lemma 6 that every maximal row in R has a nonnegative sum and that every maximal block has a nonnegative sum. We would like to characterize other submatrices of R whose sum is necessarily nonnegative.

LEMMA 7. *Consider any two maximal blocks* $B = [\ ]_{i,j}^{s,t}$ *and* $B' = [\ ]_{i',j'}^{s',t'}$, *where* $i \le i' \le i+s-1$, $i'+s' \le i+s$. *That is,* B *has height* s *and* B' *has height* $s' \le s$, *and* B' *starts at row* $i' \ge i$ *and ends at row* $i'+s'-1 \le i+s-1$. *Consider the submatrix* T *of height* s *"between them." That is,* $T = [\ ]_{i,j+t}^{s,j'-(j+t)}$ *or* $T = [\ ]_{i,j'+t'}^{s,j-(j'+t')}$. *Suppose that* $T \subset R$. *Then* $sum(T) \ge 0$.

See Figure 6 for a illustration of the lemma and its proof.

Proof. Assume without loss of generality that B' is to the right of B (that is, $j' \ge j+t$ and $T = [\ ]_{i,j+t}^{s,j'-(j+t)}$). If T is empty, then the claim follows trivially since $sum(T) = 0$. Hence we may assume from now on that T is not empty, and we separate the proof into two cases.

Case 1. T *is a legal submatrix.* Assume, contrary to the claim, that $sum(T) < 0$. That is, T is a bad legal submatrix. Let T' be the maximal bad legal submatrix containing T (where T' may equal T). By construction of R, R should contain a good legal submatrix T'' that contains T' and has twice the number of rows and twice the

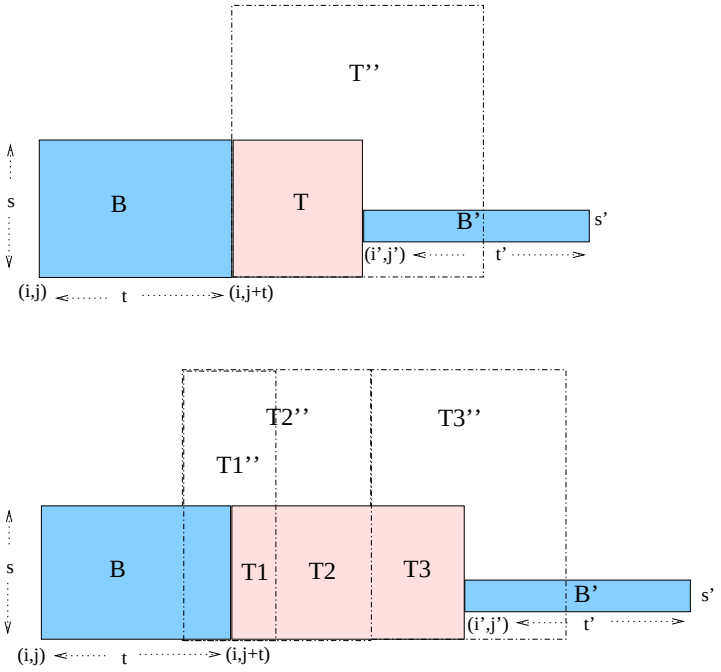


FIG. 6. *An illustration for Lemma* 7. *The figure on the top illustrates the case in the proof of Lemma* 7, *where* T *is a legal submatrix (for simplicity, we assume* $T' = T$*). The figure on the bottom illustrates the second case in the proof when* T *is a union of legal submatrices (all having the height of* B*).*

number of columns. But this would contradict the maximality of the subcolumns of B or of B'. To see why this is true, assume without loss of generality that for any legal subcolumn $[\,]^{s,1}_{i,r}$, the legal column that is twice its height is $[\,]^{2s,1}_{i,r}$ (the case in which it is $[\,]^{2s,1}_{i-s,r}$, is treated analogously). Then T'' must contain either the subcolumn $[\,]^{2s,1}_{i,j'}$ or the subcolumn $[\,]^{2s,1}_{i,j+t-1}$ (depending on the identity of the legal subrows that are twice the length of the rows of T). In the first case we would get a contradiction to the fact that B' is a maximal block, and in the second case we would get a contradiction to the fact that B is a maximal block.

Case 2. T *is not a legal submatrix.* Observe that its columns are necessarily legal subcolumns (given that the columns of B are legal). Hence, only its rows are not legal subrows. Therefore, T can be partitioned into submatrices $T_1, \ldots, T_k$ such that each is of height s and is a maximal legal submatrix *with respect to* T. We claim that for every T_ℓ, $sum(T_\ell) \geq 0$. Consider any fixed T_ℓ. By its maximality with respect to T, we know that the legal subrows that contain the rows of T_ℓ and are twice their length are not strictly contained in T, but rather they extend either to the right or to the left of T. Hence these rows (or some of them in case the height of B' is strictly smaller than the height of T_ℓ) must intersect either B or B'. Assume, contrary to what we claim, that $sum(T_\ell) < 0$. Let T'_ℓ be the maximal bad legal submatrix with respect to R that contains T_ℓ, and let T''_ℓ be the good legal submatrix that contains T'_ℓ and has twice its height and twice its width. Then T''_ℓ intersects either B or B', and in this intersection, the (legal) subcolumns of T''_ℓ strictly contain the subcolumns of B or B' (as in the case considered in the previous paragraph). But this contradicts the

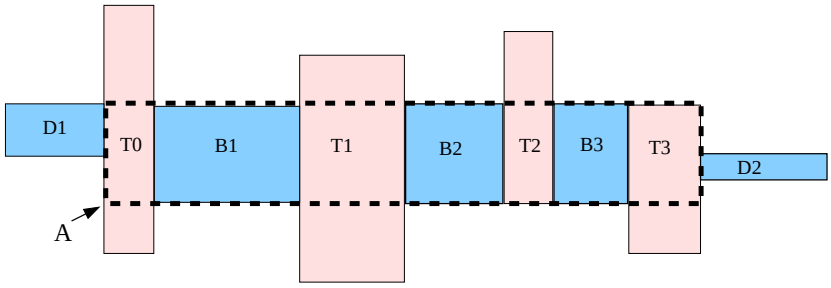


FIG. 7. *An illustration for Corollary* 8. *Here A covers the blocks B_1, B_2, and B_3 and borders the blocks D_1 and D_2. The submatrices T_0–T_4 are parts of larger blocks (that extend above and/or below A).*

maximality of B or B'. □

By Lemma 7, we get the following corollary whose proof is illustrated in Figure 7.

COROLLARY 8. *Let A be a submatrix of R that covers a given block B. If on each of its sides A either borders a block with height smaller than the height of B or its border coincides with the border of R, then* $sum(A) \geq sum(B)$.

Proof. Let $B_1, \ldots, B_k$ be the set of maximal blocks that are covered by A (where $B = B_i$ for some $1 \leq i \leq k$). Note that by definition of maximal blocks and covers, they are all of the same height, which is the height of A. Let D_1 and D_2 be two shorter blocks that border A on the left side and the right side of A, respectively. (If there is no such block on one of the sides, then we think of the corresponding D_i as having height 0.) Let $T_0, \ldots, T_k$ be the submatrices between these blocks (that have the same height as the blocks). That is, T_0 is between D_1 and B_1, T_k is between B_k and D_2, and for $1 \leq i \leq k-1$, T_i is between B_i and B_{i+1}. Then, by Lemma 7 and the fact that every block has a nonnegative sum we get that

$$sum(A) = \sum_{i=1}^{k} sum(B_i) + \sum_{i=0}^{k} sum(T_i) \geq sum(B). \qquad (6)$$ □

5.4.2. Proving that Procedure 3 does not get stuck. Recall that for each $1 \leq p \leq m$, R_p is what remains of R at the start of the pth iteration of Procedure 3. In particular, $R_1 = R$. In this section we show that the procedure does not "get stuck." That is, for each iteration p, Procedure 2 can be applied to the block B_p selected in this iteration, and it is possible to update the designated sums of the rows that have been shortened by the removal of B_p. Note that since the blocks are selected according to increasing (nondecreasing) height, then in each iteration there indeed exists a unique cover A_p of B_p that is a maximal row-cover with respect to R_p.

For every $1 \leq p \leq m$, let s_p be the minimum height of the maximal blocks of R_p, and let $s_0 = 1$. Observe that whenever s_p increases, it does so by a factor of 2^k for some k. This is true because the columns of maximal blocks are legal subcolumns.

LEMMA 9. *For every $1 \leq p \leq m$, Procedure 2 can be applied to the block B_p selected in R_p, and the updating process of the designated sum of rows can be applied. Moreover, if A is a submatrix of R_p with height of at least s_{p-1} whose columns are legal subcolumns and whose rows are maximal rows with respect to R_p, then* $\sum_{row\ L \in A} \overline{sum}(L) = sum(A)$.

Proof. Let B_p be the block selected in iteration p, where B_p is an $s \times t$ submatrix, and let A_p be the maximal row-cover of B_p with respect to R_p. As noted in

subsection 3.1, all that is required for Procedure 2 to work is the following:

(1) For every column K in B_p, $sum(K) \geq 0$.
(2) For every row L in A_p, $\overline{sum}(L) \geq 0$.
(3) $\sum_{\text{row } L \in A_p} \overline{sum}(L) \geq \sum_{\text{column } K \in B_p} sum(K)$.

In order for the updating process to succeed in step 3 of Procedure 3, we must have the following:

(4) For each $1 \leq \ell \leq s$, let x_ℓ be the sum of elements filled in the ℓth subrow of B_p, and let L_ℓ be the subrow of A_p that covers this subrow of B_p. Then $\overline{sum}(L_\ell) - x_\ell \geq 0$.
(5) If $A_p \setminus B_p$ consists of the two submatrices A' and A'' (between which resided B), then $sum(A') \geq 0$, $sum(A'') \geq 0$, and

$$\sum_{\text{row } L_\ell \in A_p} (\overline{sum}(L_\ell) - x_\ell) = sum(A') + sum(A'').$$

By Lemma 6, item (1) holds at the start of every iteration. In order to prove the other items for every p, we first extend and generalize item (2):

(2') Let A be any submatrix in R_p having height at least s_{p-1} whose columns are legal subcolumns and whose rows are maximal rows with respect to R_p. Then for every row L of A we have $\overline{sum}(L) \geq 0$, and $\sum_{\text{row } L \in A} \overline{sum}(L) = sum(A)$.

Observe that if item (2') holds at the start of iteration p, then in particular it holds for A_p. Hence by Corollary 8

$$\sum_{\text{row } L \in A_p} \overline{sum}(L) = sum(A_p) \geq sum(B_p) \tag{7}$$

and so item (3) holds as well.

Furthermore, if items (1)–(3) hold at the start of iteration p, then Procedure 2 can be applied successfully. Thus item (4) necessarily holds by definition of Procedure 2. The first part of item (5), concerning the nonnegativity of A' and A'', follows from Lemma 7 very similarly to the way Corollary 8 follows from this lemma. The second part of item (5) follows from item (2') holding for A_p and the fact that $\sum_{\ell=1}^{s} x_\ell = sum(B_p)$ (since Procedure 2 completed successfully). Hence,

$$\sum_{\text{row } L_\ell \in A_p} (\overline{sum}(L_\ell) - x_\ell) = sum(A_p) - sum(B_p) = sum(A') + sum(A'') \tag{8}$$

as required.

Hence, it remains to prove that item (2') holds at the start of every iteration p. We do so by induction on p. Consider the base case, $p = 1$, so that $R_p = R_1 = R$. By the initialization of Procedure 3, for every maximal subrow L of R, $\overline{sum}(L) = sum(L)$. By Lemma 6 (applied to the maximal legal subrows that partition L), we know that $\overline{sum}(L) \geq 0$. Furthermore, for every submatrix A of R having height of at least $s_{p-1} = s_0 = 1$ and whose rows are maximal subrows of R,

$$\sum_{\text{row } L \in A} \overline{sum}(L) = \sum_{\text{row } L \in A} sum(L) = sum(A) \tag{9}$$

as required.

Assuming that the induction claim holds for $p - 1$, we prove it for p. Consider any submatrix A having height at least s_{p-1} whose columns are legal subcolumns and

whose rows are maximal subrows with respect to R_p. If A also consisted of maximal subrows with respect to R_{p-1}, then we are done by the induction hypothesis.

Otherwise, the block B_{p-1} of height s_{p-1} that was removed from R_{p-1} bordered A on one of its sides. Let $A^1, \ldots, A^q$ be the disjoint submatrices of height s_{p-1} such that $A = \cup_{h=1}^{q} A^h$. That is, $A^1, \ldots, A^q$ are located one on top of the other (for an illustration, see Figure 8). In this case, all but at most one of these submatrices, say A^q, consisted of maximal subrows with respect to R_{p-1}, and B_{p-1} bordered A^q.

For each of the submatrices $A^1, \ldots, A^{q-1}$ we can apply the induction hypothesis (item (2')). We get the following for each such A^h: (a) for every row L in A^h, $\overline{sum}(L) \geq 0$; and (b) $\sum_{\text{row } L \in A^h} \overline{sum}(L) = sum(A^h)$.

As for A^q, assume without loss of generality that B_{p-1} bordered A^q from the right of A^q. Let A' be the submatrix that bordered B_{p-1} from the right of B_{p-1} (A' may be empty). This means that A_{p-1} is of the form $A_{p-1} = A^q \cup B_{p-1} \cup A'$ (see Figure 8). But then, by definition of the updating rule and since it succeeded by the induction hypothesis (items (4) and (5)), we have that for every row L in A^q, $\overline{sum}(L) \geq 0$ and $\sum_{\text{row } L \in A^q} \overline{sum}(L) = sum(A^q)$.

It follows that for every row L in A we have $\overline{sum}(L) \geq 0$ and

$$\sum_{\text{row } L \in A} \overline{sum}(L) = \sum_{h=1}^{q} \sum_{\text{row} L \in A^h} \overline{sum}(L) = \sum_{h=1}^{q} sum(A^h) = sum(A) \,. \tag{10}$$

The induction step is proven. □

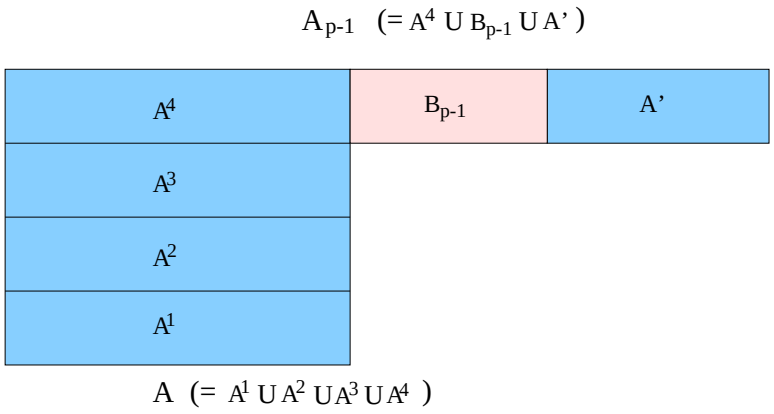


FIG. 8. *An illustration for the induction step in the proof of Lemma* 9 *(where* $q = 4$*).*

5.4.3. Proving that Property 1 holds at the end of Procedure 3. Finally, we have to show that when Procedure 3 terminates and R is refilled with nonnegative values, then Property 1 holds. This will complete the proof of Theorem 3.

Let $\tilde{C}_V = \{\tilde{c}_{i,j}\}$ be the matrix resulting from the application of Procedure 3 to the matrix $C_V = \{c_{i,j}\}$. For any submatrix T of C_V (and in particular of R), we let $\widetilde{sum}(T)$ denote the sum of elements of T in $\tilde{C}_V$. By definition of the procedure, $\widetilde{sum}(K) = sum(K)$ for every maximal legal subcolumn K of R. Hence this holds also for every maximal subcolumn of R. We next prove a related claim concerning rows.

LEMMA 10. *For every subrow* L *in* R *such that* L *is assigned* $\overline{sum}(L)$ *as a designated sum at some iteration of Procedure* 3*, we have that* $\widetilde{sum}(L) = \overline{sum}(L)$.

Observe that by combining Lemma 10 with Lemma 6 we get that for every maximal subrow L of R, $\widetilde{sum}(L) = \overline{sum}(L) = sum(L)$.

Proof. Let $\mathcal{L}$ be the set of subrows L of R such that L is assigned $\overline{sum}(L)$ as a designated sum at some iteration of Procedure 3. Observe that the set $\mathcal{L}$ consists exactly of those rows that are maximal subrows for some R_p. We prove the lemma by induction on the length of $L \in \mathcal{L}$. For the base of the induction, consider any subrow $L \in \mathcal{L}$ that is shortest among all subrows in $\mathcal{L}$. Since L is shortest, it must be completely filled in a single iteration as part of a block B (or otherwise there would be a shorter $L' \subset L$ with a designated sum $\overline{sum}(L')$). But by definition of the procedure, we get that $\widetilde{sum}(L) = \overline{sum}(L)$ as required.

Assume that the claim holds for every L of length less than ℓ; we prove it for L having length ℓ. Consider the first iteration after which L became a maximal subrow (and thus received the designated sum $\overline{sum}(L)$) in which part of L is filled. If all of L is filled, then the induction claim follows as in the base case. Otherwise, let x be the sum of elements that was filled in the part $P \subset L$. Let L' and L'' be what remains of L to the left and right of P, respectively. Then the procedure sets $\overline{sum}(L') + \overline{sum}(L'') = \overline{sum}(L) - x$. But L' and L'' are strictly shorter than L, and therefore by the induction hypothesis $\widetilde{sum}(L') = \overline{sum}(L')$ and $\widetilde{sum}(L'') = \overline{sum}(L'')$. Thus $\widetilde{sum}(L) = \widetilde{sum}(L') + \widetilde{sum}(L'') + x = \overline{sum}(L') + \overline{sum}(L'') + x = \overline{sum}(L)$ as required. ∎

Definition 16 (boundary). *We say that a point (i,j) is on the* boundary *of R if $(i,j) \in R$, but either $(i+1,j) \notin R$, or $(i,j+1) \notin R$, or $(i+1,j+1) \notin R$. We denote the set of boundary points by $\mathcal{B}$.*

Definition 17. *For a point (i,j), $1 \le i,j \le n$, let $R^{\le}(i,j)$ denote the subset of points $(i',j') \in R$, $i' \le i, j' \le j$, and let $sum^R(i,j) = \sum_{(i',j') \in R^{\le}(i,j)} c_{i',j'}$ and $\widetilde{sum}^R(i,j) = \sum_{(i',j') \in R^{\le}(i,j)} \tilde{c}_{i',j'}$.*

Property 1 and therefore Theorem 3 will follow directly from the next two lemmas.

Lemma 11. *For every point $(i,j) \in \mathcal{B}$, $\widetilde{sum}^R(i,j) = sum^R(i,j)$.*

Proof. Consider any point $(i,j) \in \mathcal{B}$, and let $U = R^{\le}(i,j)$. Let $\mathcal{C}(U) = \{B_1, \ldots, B_q\}$ be the minimal set of (maximal) blocks whose union contains U. For each $B_h \in \mathcal{C}(U)$ we know that $\widetilde{sum}(B_h) = sum(B_h)$. In particular, this is true for every $B_h \subset U$. Let $\mathcal{C}_1(U) = \{B_h \in \mathcal{C}(U) : B_h \subset U\}$. Hence we have that

$$\sum_{B_h \in \mathcal{C}_1(U)} \widetilde{sum}(B_h \cap U) = \sum_{B_h \in \mathcal{C}_1(U)} \widetilde{sum}(B_h) = \sum_{B_h \in \mathcal{C}_1(U)} sum(B_h)\,. \tag{11}$$

If every $B_h \in \mathcal{C}(U)$ is fully contained in U, then $\mathcal{C}_1(U) = \mathcal{C}(U)$ and we are done.

Otherwise, consider the remaining B_h's in $\mathcal{C}(U) \setminus \mathcal{C}_1(U)$ (i.e., blocks that are not fully contained in U but rather intersect it). Each of them contains either a column that is a subcolumn of column $j+1$ or a row that is a subrow of row $i+1$ (recall that $U = R^{\le}(i,j)$). Let the former subset be denoted $\mathcal{C}_2(U)$ and the latter $\mathcal{C}_3(U)$. Thus $\mathcal{C}_2(U)$ contains blocks that "intersect U from the right," and $\mathcal{C}_3(U)$ contain blocks that "intersect U from the top." See, for example, Figure 9.

It is important to note that $\mathcal{C}_2(U) \cap \mathcal{C}_3(U) = \emptyset$: If there existed a block $B_h \in \mathcal{C}_2(U) \cap \mathcal{C}_2(U)$, it would necessarily contain both (i,j) and the three neighboring points, $(i+1,j)$, $(i,j+1)$, and $(i+1,j+1)$. But this contradicts the fact that (i,j) is a boundary point.

For each $B_h \in \mathcal{C}_2(U)$, $B_h \cap U$ is a subset of maximal legal subcolumns with respect to R (since each $B_h \in \mathcal{C}_2(U)$ cannot extend beyond row i). Let $\mathcal{K}_2(U)$ denote the set

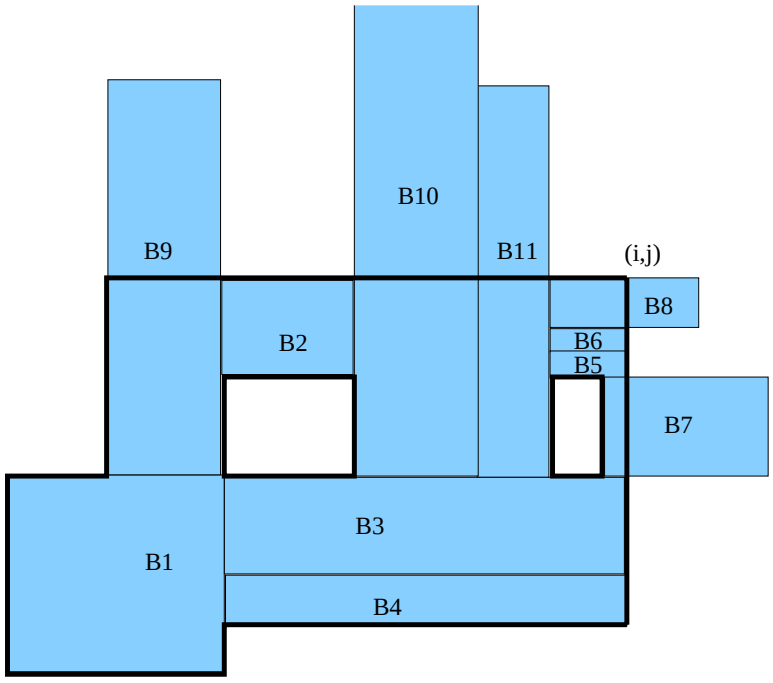


FIG. 9. *An illustration for the proof of Lemma* 11. *The solid line denotes the outline of* $U = R^{\leq}(i,j)$, *where point* (i,j) *is in the top-right corner. Blocks* B_1–B_6 *are fully contained in* U *and therefore belong to* $\mathcal{C}_1(U)$. *Blocks* B_7 *and* B_8 *belong to* $\mathcal{C}_2(U)$ *and blocks* B_9–B_{11} *belong to* $\mathcal{C}_3(U)$. *Block* B_{10} *is twice the height of* B_9 *and* B_{11} *and thus "extends out of the figure."*

of all maximal legal subcolumns that belong to $\bigcup_{B_h \in \mathcal{C}_2(U)}(B_h \cap U)$. Since for every maximal legal subcolumn K it holds that $\widetilde{sum}(K) = sum(K)$, we have that

$$\sum_{B_h \in \mathcal{C}_2(U)} \widetilde{sum}(B_h \cap U) = \sum_{K \in \mathcal{K}_2(U)} \widetilde{sum}(K) = \sum_{K \in \mathcal{K}_2(U)} sum(K). \tag{12}$$

Next consider the blocks $B_h \in \mathcal{C}_3(U)$. Let $\mathcal{L}_3(U)$ be the set of subrows in U that are maximal subrows with respect to $\bigcup_{B_h \in \mathcal{C}_3(U)}(B_h \cap U)$. Thus, $\bigcup_{B_h \in \mathcal{C}_3(U)}(B_h \cap U) = \bigcup_{L \in \mathcal{L}_3} L$. We next observe that for every $B_h \in \mathcal{C}_3(U)$, all blocks that border B_h and belong either to $\mathcal{C}_1(U)$ or to $\mathcal{C}_2(U)$ must be strictly shorter than B_h. This follows from the definition of legal subcolumns. Hence, the blocks in $\mathcal{C}_1(U)$ and $\mathcal{C}_2(U)$ are all removed before the blocks in $\mathcal{C}_3(U)$.

For each subrow in $\mathcal{L}_3(U)$ there exists the first iteration p in which it becomes a maximal subrow with respect to R_p (following the removal of some block in $\mathcal{C}_1(U) \cup \mathcal{C}_2(U)$ from R_{p-1}). We partition the rows in $\mathcal{L}_3(U)$ accordingly. Let $\mathcal{L}_3^p(U)$ denote all subrows in $\mathcal{L}_3(U)$ that are maximal subrows with respect to R_p but were not maximal subrows with respect to R_{p-1}. Observe that, in particular, $\mathcal{L}_3^1(U)$ is the set of subrows in $\mathcal{L}_3(U)$ that were already maximal subrows with respect to R. By this definition the subrows in $\mathcal{L}_3^p(U)$ constitute a submatrix of height s_{p-1}. By the second part of Lemma 9, $\sum_{L \in \mathcal{L}_3^p(U)} \widetilde{sum}(L) = \sum_{L \in \mathcal{L}_3^p(U)} sum(L)$, and by applying Lemma 10 we

get that $\sum_{L\in\mathcal{L}_3(U)} \widetilde{sum}(L) = \sum_{L\in\mathcal{L}_3(U)} sum(L)$. Therefore,

$$\sum_{B_h\in\mathcal{C}_3(U)} \widetilde{sum}(B_h\cap U) = \sum_{L\in\mathcal{L}_3(U)} \widetilde{sum}(L) = \sum_{L\in\mathcal{L}_3(U)} sum(L). \tag{13}$$

By combining (11)–(13) we get

$$\begin{aligned}\widetilde{sum}(U) &= \sum_{B_h\in\mathcal{C}(U)} \widetilde{sum}(B_h\cap U)\\ &= \sum_{q=1}^{3}\sum_{B_h\in\mathcal{C}_q(U)} \widetilde{sum}(B_h\cap U)\\ &= \sum_{B_h\in\mathcal{C}_1(U)} sum(B_h) + \sum_{K\in\mathcal{K}_2(U)} sum(K) + \sum_{L\in\mathcal{L}_3(U)} sum(L)\\ &= sum(U) \qquad \square\end{aligned}$$

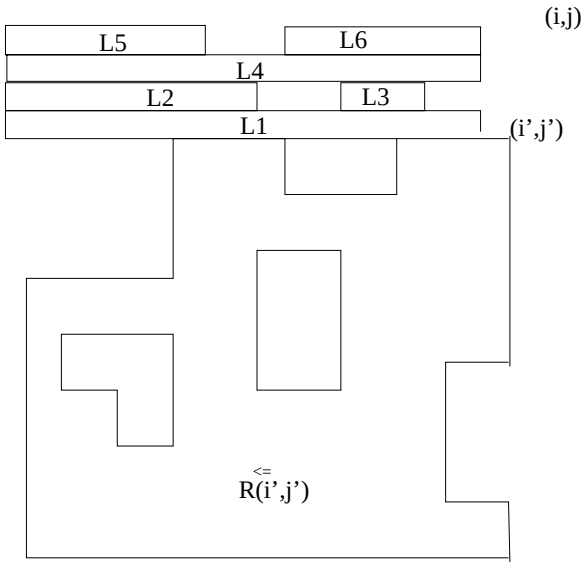


FIG. 10. *An illustration for the proof of Lemma* 12. *The point* (i', j') *is as defined in the proof, and the rows* $L_1, \ldots, L_6$ *are all maximal subrows of* R *that belong to rows* $i'+1, \ldots, i$ *and end by column* j *(that is, the set* $\mathcal{L}(i, i', j)$*).*

LEMMA 12. *Let* (i, j) *be any point such that* $(i, j) \notin R$. *Then* $\widetilde{sum}^R(i, j) = sum^R(i, j)$.

Proof. Let $(i', j') \in R$, $i' < i$, $j' \le j$, be the point for which j' is maximized, and if there are several such points, let it be the one amongst them for which i' is maximized. Thus, (i', j') is maximal in the sense that for every (i'', j''), $i'' < i$, $j'' \le j$ such that $(i'', j'') > (i', j')$ it holds that $(i'', j'') \notin R$. Furthermore, among all such maximal points it is the right-most one (i.e., it belongs to the column with the highest index). By definition, (i', j') belongs to $\mathcal{B}$, since $(i'+1, j')$ necessarily does not belong to R. Let $\mathcal{L}(i, i', j)$ be the subset of all maximal subrows of R that belong to rows $i'+1, \ldots, i$ and end by column j. Then $\widetilde{sum}^R(i, j) = \widetilde{sum}^R(i', j') + \sum_{L\in\mathcal{L}(i,i',j)} \widetilde{sum}(L)$. By

applying Lemma 11 and Lemma 10, we get that $\widetilde{sum}^R(i,j) = sum^R(i,j)$. For an illustration, see Figure 10. □

5.5. Distribution matrices. As noted in the introduction, a subfamily of inverse Monge matrices that is of particular interest is the class of *distribution matrices*. A matrix $V = \{v_{i,j}\}$ is said to be a distribution matrix if there exists a nonnegative *density matrix* $D = \{d_{i,j}\}$ such that every entry $v_{i,j}$ in V is of the form $v_{i,j} = \sum_{k \le i} \sum_{\ell \le j} d_{k,\ell}$. In particular, if V is a distribution matrix, then the corresponding density matrix D is simply the matrix C'_V (as defined in section 3). Hence, in order to test that V is a distribution matrix, we simply run our algorithm for inverse Monge matrix on C'_V instead of C_V.

Acknowledgments. We would like to thank Noam Nisan for suggesting to examine combinatorial auctions in the context of property testing. We would also like to thank the anonymous referees for their comments which helped us improve the presentation of this paper.

REFERENCES

[BKR96] R. E. Burkard, B. Klinz, and R. Rudolf, *Perspectives of Monge properties in optimization*, Discrete Appl. Math., 70 (1996), pp. 95–161.

[BRW99] T. Batu, R. Rubinfeld, and P. White, *Fast approximate PCPs for multidimensional bin-packing problems*, in Proceedings of RANDOM, Berkeley, CA, 1999, pp. 245–256.

[DGL+99] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky, *Improved testing algorithms for monotonocity*, in Proceedings of RANDOM, Berkeley, CA, 1999, pp. 97–108.

[dVV00] S. de Vries and R. Vohra, *Combinatorial Auctions: A Survey*, http://www.kellogg.nwu.edu/faculty/vohra/htm/res.htm (2000).

[EKK+00] F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan, *Spot-checkers*, J. Comput. System Sci., 60 (2000), pp. 717–751.

[Fis01] E. Fischer, *The art of uninformed decisions: A primer to property testing*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 75 (2001), pp. 97–126.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.

[GLS81] M. Grötschel, L. Lovász, and A. Schrijver, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, 1 (1981), pp. 169–197.

[Hof63] A. J. Hoffman, *On Simple Linear Programming Problems*, Proc. Sympos. Pure Math. 7, AMS, Providence, RI, 1963, pp. 317–327.

[IFF01] S. Iwata, L. Fleischer, and S. Fujishige, *A combinatorial strongly polynomial algorithm for minimizing submodular functions*, J. ACM, 48 (2001), pp. 761–777.

[LLN01] B. Lehmann, D. Lehmann, and N. Nisan, *Combinatorial auctions with decreasing marginal utilities*, in Proceedings of the ACM Conference on Electronic Commerce, Tampa, FL, 2001.

[Lov83] L. Lovász, *Submodular functions and convexity*, Mathematical Programming: The State of the Art, 1983, Springer, Berlin, pp. 235–257.

[Ron01] D. Ron, *Property testing*, in Handbook on Randomization, Vol. II, S. Rajasekaran, P. Pardalos, J. Reif, and J. Rolim, eds., Kluwer Academic Publishers, Dordrecht, 2001, pp. 597–649.

[RS96] R. Rubinfeld and M. Sudan, *Robust characterization of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.

[Sch00] A. Schrijver, *A combinatorial algorithm minimizing submodular functions in strongly polynomial time*, J. Combin. Theory Ser. B, 80 (2000), pp. 346–355.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1185–1200

LOWER BOUNDS FOR MATRIX PRODUCT*

AMIR SHPILKA†

Abstract. We prove lower bounds on the number of product gates in bilinear and quadratic circuits that compute the product of two $n \times n$ matrices over finite fields. In particular we obtain the following results:

1. We show that the number of product gates in any *bilinear* (or *quadratic*) circuit that computes the product of two $n \times n$ matrices over GF(2) is at least $3n^2 - o(n^2)$.
2. We show that the number of product gates in any *bilinear* circuit that computes the product of two $n \times n$ matrices over GF(q) is at least $(2.5 + \frac{1.5}{q^3-1})n^2 - o(n^2)$.

These results improve the former results of [N. H. Bshouty, *SIAM J. Comput.*, 18 (1989), pp. 759–765; M. Bläser, *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 45–50], who proved lower bounds of $2.5n^2 - o(n^2)$.

Key words. matrix product, lower bounds, linear codes

AMS subject classification. 68Q17

DOI. 10.1137/S0097539702405954

1. Introduction.

The problem of computing the product of two matrices is one of the most studied computational problems. We are given two $n \times n$ matrices $x = (x_{i,j})$, $y = (y_{i,j})$, and we wish to compute their product; i.e., there are n^2 outputs, where the (i,j)th output is

$$(x \cdot y)_{i,j} = \sum_{k=1}^{n} x_{i,k} \cdot y_{k,j}.$$

In 1969, Strassen surprised the world by showing an upper bound of $O(n^{\log_2 7})$ [Str69]. This bound was later improved, and the best upper bound today is $O(n^{2.376})$ [CW90] (see also [Gat88] for a survey). The best known lower bound, $2.5n^2 - o(n^2)$, on the number of products needed to compute the function is due to [Bsh89, Bla99]. Thus the following problem is still open: Can matrix products be computed by a circuit of size $O(n^2)$?

The standard computational model for computing polynomials is the model of arithmetic circuits, i.e., circuits over the base $\{+, \cdot\}$ over some field F. This is indeed the most general model, but for matrix products two other models are usually considered, *quadratic circuits* and *bilinear circuits*. In the quadratic model we require that product gates be applied on only two linear functions. In the bilinear model we also require that product gates be applied on only two linear functions, but in addition we require that the first linear function be linear in the variables of x and that the second linear function be linear in the variables of y. These models are more restricted than the general model of arithmetic circuits. However, it is interesting to note that over infinite fields we can always assume w.l.o.g. that any circuit for a matrix product is a quadratic circuit [Str73]. In addition we note that the best circuits that

*Received by the editors January 2, 2002; accepted for publication (in revised form) April 29, 2003; published electronically August 6, 2003.
http://www.siam.org/journals/sicomp/32-5/40595.html
†Division of Engineering and Applied Sciences, Harvard University, and Laboratory for Computer Science, M.I.T., Cambridge, MA 02138 (amirs@deas.harvard.edu).

we have today for matrix products are bilinear circuits (except for very small values of n, where the best circuits are quadratic circuits; see [Win68, Wak70, BCS97]).

In this paper we prove that any *quadratic* circuit that computes matrix products over the field GF(2) has at least $3n^2-o(n^2)$ product gates, and that any *bilinear* circuit for matrix products over the field GF(q) must have at least $(2.5+\frac{1.5}{q^3-1})n^2-o(n^2)$ product gates.

From now on we will use the notation MP_n to denote the problem of computing the product of two $n\times n$ matrices.

1.1. Known lower bounds. In contrast to the major advances in proving upper bounds, the attempts to prove lower bounds on the size of bilinear circuits that compute MP_n were less successful. Denote by $\mathrm{q}_*^{\mathrm{F}}(\mathrm{MP}_n)$ and $\mathrm{bl}_*^{\mathrm{F}}(\mathrm{MP}_n)$ the smallest number of product gates in a quadratic circuit for MP_n, and in a bilinear circuit for MP_n, respectively, over a field F. We also denote by $\mathrm{bl}_{\mathrm{tot}}^{\mathrm{F}}(\mathrm{MP}_n)$ the total number of gates in a smallest bilinear circuit for MP_n. In 1978, Brockett and Dobkin proved that $\mathrm{bl}_*^{\mathrm{F}}(\mathrm{MP}_n)\geq 2n^2-1$ over any field F (see [BD78]). This lower bound was later generalized by Lafon and Winograd to a lower bound on $\mathrm{q}_*^{\mathrm{F}}(\mathrm{MP}_n)$ over any field [LW78]. In 1989, Bshouty showed that $\mathrm{q}_*^{\mathrm{GF}(2)}(\mathrm{MP}_n)\geq 2.5n^2-O(n\log n)$ [Bsh89]. Recently, in a series of works, Bläser managed to prove a lower bound of $2.5n^2-3n$ on $\mathrm{q}_*^{\mathrm{F}}(\mathrm{MP}_n)$ over any field [Bla99, Bla00, Bla01].

In [RS01] it is shown that any bounded depth circuit for MP_n, over any field, has a superlinear (in n^2) size. Notice, however, that the best known circuits for MP_n have depth $\Omega(\log n)$.

1.2. Bilinear rank. An important notion that is highly related to the problem of computing matrix product in bilinear circuits is the notion of bilinear rank.

A bilinear form in two sets of variables x,y is a polynomial in the variables of x and in the variables of y, which is linear in the variables of x and linear in the variables of y. Clearly each output of MP_n is a bilinear form in $x=\{x_{i,j}\}$, $y=\{y_{i,j}\}$. The bilinear rank of a set of bilinear forms $\{b_1(x,y),\ldots,b_m(x,y)\}$ is the smallest number of rank-1 bilinear forms that span $b_1,\ldots,b_m$, where a rank-1 bilinear form is a product of a linear form in the x variables and a linear form in the y variables. We denote by $\mathrm{R}_{\mathrm{F}}(b_1,\ldots,b_m)$ the bilinear rank of $\{b_1,\ldots,b_m\}$ over the field F. For further background see [BCS97, Gat88].

We denote by $\mathrm{R}_{\mathrm{F}}(\mathrm{MP}_n)$ the bilinear rank over F of the n^2 outputs of the matrix product; i.e., it is the bilinear rank of the set $\{\sum_{k=1}^{n}x_{i,k}\cdot y_{k,j}\}_{i,j}$ over F.

The following inequalities are obvious (over any field):

- $\mathrm{q}_*^{\mathrm{F}}(\mathrm{MP}_n)\leq \mathrm{bl}_*^{\mathrm{F}}(\mathrm{MP}_n)\leq 2\mathrm{q}_*^{\mathrm{F}}(\mathrm{MP}_n)$.
- $\mathrm{R}_{\mathrm{F}}(\mathrm{MP}_n)=\mathrm{bl}_*^{\mathrm{F}}(\mathrm{MP}_n)$.

The following statement is less obvious, but also not so hard to see. When n grows to infinity, $\mathrm{bl}_*^{\mathrm{F}}(\mathrm{MP}_n)$ and $\mathrm{bl}_{\mathrm{tot}}^{\mathrm{F}}(\mathrm{MP}_n)$ are (roughly) of the same order of magnitude. This statement is formulated and proved in the appendix.

1.3. Results and methods. We prove that any *quadratic* circuit that computes MP_n over the field GF(2) has at least $3n^2-o(n^2)$ product gates (i.e., $\mathrm{q}_*^{\mathrm{GF}(2)}(\mathrm{MP}_n)\geq 3n^2-o(n^2)$). We also prove that over the field GF(q) every *bilinear* circuit for MP_n must have at least $(2.5+\frac{1.5}{q^3-1})n^2-o(n^2)$ product gates (i.e., $\mathrm{bl}_*^{\mathrm{GF}(q)}(\mathrm{MP}_n)\geq (2.5+\frac{1.5}{q^3-1})n^2-o(n^2)$). Both of these results actually hold for the bilinear rank as well.

The proof of the lower bound over GF(2) is based on techniques from the theory of linear codes. However, we cannot use known results from coding theory in a

straightforward way, since we are not dealing with codes in which every two words are distant, but rather with codes on matrices in which the distance between two code words, of two matrices, is proportional to the rank of the difference of the matrices. The reduction from circuits to codes and the proof of the bound are given in section 4.

The proof of the second bound is based on a lemma proved by Bläser in [Bla99]. We prove that in the case of finite fields we can use the lemma with better parameters than those used by Bläser. This result is proved in section 5.

1.4. Organization of the paper. In section 2 we present the models of bilinear circuits and quadratic circuits. In section 3 we present some algebraic and combinatorial tools that we need for the proofs of our lower bounds.

In section 4 we introduce the notion of linear codes of matrices and prove our lower bound on bilinear and quadratic circuits that compute MP_n over $\mathrm{GF}(2)$. In section 5 we prove our lower bound on bilinear circuits that compute MP_n over $\mathrm{GF}(q)$.

Finally we discuss the limits of our techniques in section 6.

2. Arithmetic models. In this section we present the models of quadratic circuits and bilinear circuits. These are the models for which we prove our lower bounds. We first give the definition of a general arithmetic circuit. An arithmetic circuit over a field F is a directed acyclic graph as follows. Nodes of in-degree 0 are called inputs and are labeled with input variables. Nodes of out-degree 0 are called outputs. Each edge is labeled with a constant from the field, and each node other than an input is labeled with one of the operations $\{ + ; \cdot \}$; in the first case the node is a plus gate and in the second case a product gate. The computation is done in the following way. An input just computes the value of the variable that labels it. Then, if $v_1, \ldots, v_k$ are the vertices that fan into v, we multiply the result of each v_i with the value of the edge that connects it to v. If v is a plus gate, we sum all the results; otherwise v is a product gate, and we multiply all the results. Obviously the value computed by each node in the circuit is a polynomial over F in the input variables.

We are interested in the problem of computing the product of two $n \times n$ matrices, MP_n. The input consists of two $n \times n$ matrices x, y. The output is the matrix $x \cdot y$, i.e., there are n^2 outputs, and the (i,j)th output is

$$(x \cdot y)_{i,j} = \sum_{k=1}^{n} x_{i,k} \cdot y_{k,j}.$$

Each output $(x \cdot y)_{i,j}$ is hence a bilinear form in x and y.

Since each output of MP_n is a bilinear form, it is natural to consider bilinear arithmetic circuits for it. A bilinear arithmetic circuit is an arithmetic circuit with the additional restriction that product gates are applied on only two linear functions, one function linear in the variables of x and the other function linear in the variables of y. Thus, bilinear circuits have the following structure. First, there are many plus gates computing linear forms in x and linear forms in y. Then there is one level of product gates that compute bilinear forms, and finally there are many plus gates that eventually compute the outputs. We will be interested in bounding from below the number of products in any bilinear circuit for MP_n. This model is more restricted than the general model of arithmetic circuits, but we note that all the known upper bounds (over any field) for MP_n are achieved using bilinear circuits.

Another model that we will consider is the model of quadratic circuits. A quadratic circuit is an arithmetic circuit with the additional restriction that product gates are applied on only two linear functions. Notice that the only difference between quadratic

circuits and bilinear circuits is that in the quadratic model the product gates compute *quadratic forms* in x, y, whereas in the bilinear model the product gates compute *bilinear forms* in x, y. This model is more general than the model of bilinear circuits, but it is still more restricted than the general model. However, it is interesting to note that over infinite fields we can assume w.l.o.g. that any arithmetic circuit for MP_n is a quadratic circuit [Str73].

3. Algebraic and combinatorial tools. In this section we present some algebraic and combinatorial tools that we will use.

The following lemma is a weak variant of the famous Schwartz–Zippel lemma, which shows that every nonzero polynomial (nonzero as a formal expression) over a large enough field has a nonzero assignment in the field (see [Sch80, Zip79]).

LEMMA 3.1. *Let P be a polynomial of total degree d in $x_1, \ldots, x_n$ over some field* F *such that $d < |\mathrm{F}|$ and such that at least one of the coefficients of P is not zero. Then we can find an assignment, $\rho \in \mathrm{F}^n$, to the x_i's such that $P(\rho_1, \ldots, \rho_n) \neq 0$.*

We say that two polynomials p, q in n variables are equivalent over a field F if $p(x_1, \ldots, x_n) = q(x_1, \ldots, x_n)$ for any $x_1, \ldots, x_n \in \mathrm{F}$. We define $p \equiv q$ if p and q are equivalent over F. (We omit F from the notation, as the field that we are dealing with will be clear from the context.)

LEMMA 3.2. *Let P be a polynomial of total degree d in the variables $x_1, \ldots, x_n$ over a field* F*. If $P \not\equiv 0$, then we can find an assignment, $\rho \in \mathrm{F}^n$, to the x_i's such that at most d of the ρ_i's get a nonzero value and such that $P(\rho_1, \ldots, \rho_n) \neq 0$.*

Proof. P is equal (as a function) to a polynomial $\bar{P}$ in which the degree of each variable is at most $|\mathrm{F}|-1$. We call $\bar{P}$ the reduction of P; notice that the total degree of $\bar{P}$ is at most the total degree of P. Consider some monomial M in $\bar{P}$ whose coefficient is not zero. We assign all the variables that do not appear in M to zero. The resulting polynomial (after the assignment) is a polynomial in the variables of M, which is not the zero polynomial, as it is a reduced polynomial which has a monomial with a nonzero coefficient (M, of course). Therefore according to Lemma 3.1 there is some assignment to the variables of M that gives this polynomial a nonzero value. Therefore we have found an assignment which gives nonzero values only to the variables of M (and there are at most d such variables) under which $P \neq 0$. □

The following useful lemma, which is a straightforward implication of the previous lemma, is the key lemma in most of our proofs. It deals with linear forms in n^2 variables. From now on we shall think about such linear forms as linear forms in the entries of $n \times n$ matrices.

LEMMA 3.3. *Let $\mu_1, \ldots, \mu_{n^2}$ be n^2 linearly independent linear forms in n^2 variables over some field* F*. Let P be a polynomial of total degree d in kn^2 variables over* F*; i.e., we can view P as a polynomial $P(x_1, \ldots, x_k)$ in the entries of k matrices, $x_1, \ldots, x_k$, each of size $n \times n$. Assume that $P \not\equiv 0$. Then we can find k matrices $a_1, \ldots, a_k \in \mathrm{M}_n(\mathrm{F})$ such that $P(a_1, \ldots, a_k) \neq 0$ and such that there exist $n^2 - d$ linear forms among $\mu_1, \ldots, \mu_{n^2}$ that vanish on all the a_i's.*

Proof. The idea of the proof is the following. Let $b_1, \ldots, b_{n^2}$ be the dual basis of $\mu_1, \ldots, \mu_{n^2}$, i.e., a basis of $\mathrm{M}_n(\mathrm{F})$ satisfying $\forall i, j$, $\mu_i(b_j) = \delta_{i,j}$. We wish to find k matrices $a_1, \ldots, a_k$ such that $P(a_1, \ldots, a_k) \neq 0$ and such that there exist $b_{i_1}, \ldots, b_{i_d}$ that span all of them. If we manage to find such matrices, then since the b_i's are the dual basis to the μ_i's, we will get that $n^2 - d$ of the μ_i's, vanish on $a_1, \ldots, a_k$. The way to find such matrices, which are contained in the span of a small subset of the b_i's, is based on Lemma 3.2.

Thus let $b_1, \ldots, b_{n^2}$ be the dual basis to $\mu_1, \ldots, \mu_{n^2}$; i.e., $\forall i, j$, $\mu_i(b_j) = \delta_{i,j}$. We

now change the variables of P. Let $\alpha_{i,j}$, $j = 1, \ldots, k$, $i = 1, \ldots, n^2$, be a set of kn^2 variables. Define $x_j = \sum_{i=1}^{n^2} \alpha_{i,j} b_i$. Thus we can write

$$P(x_1, \ldots, x_k) = P\left(\sum_{i=1}^{n^2} \alpha_{i,1} b_i, \ldots, \sum_{i=1}^{n^2} \alpha_{i,k} b_i\right) = Q(\alpha_{1,1}, \ldots, \alpha_{n^2,k}) .$$

Thus we have that $Q \not\equiv 0$ is a polynomial in the $\alpha_{i,j}$'s of total degree at most d. Hence, according to Lemma 3.2 there exists an assignment ρ to the $\alpha_{i,j}$'s such that at most d of them get a nonzero value and such that Q is nonzero. Define $a_j = \sum_{i=1}^{n^2} \rho_{i,j} b_i$. Clearly $P(a_1, \ldots, a_k) \neq 0$. Since at most d of the $\rho_{i,j}$'s have nonzero values, we see that there are at most d b_i's such that all the a_j's are linear combinations of them. Since the b_i's are the dual basis to $\mu_1, \ldots, \mu_{n^2}$, we get that there are at least $n^2 - d$ of the μ_i's that vanish on all the a_j's. Therefore $a_1, \ldots, a_k$ satisfy the requirements of the lemma. □

The next lemma will enable us to translate properties of matrices over large fields of characteristic p to properties of matrices (of higher dimension) over $\mathrm{GF}(q)$.

LEMMA 3.4. *There exists an injective ring homomorphism* $\phi : \mathrm{GF}(q^n) \hookrightarrow \mathrm{M}_n(\mathrm{GF}(q))$. *That is, there exists a mapping* $\phi : \mathrm{GF}(q^n) \mapsto \mathrm{M}_n(\mathrm{GF}(q))$ *such that*

- ϕ *is one-to-one.*
- $\phi(1) = I$, *where* I *is the* $n \times n$ *identity matrix.*
- ϕ *is linear, i.e.,* $\forall x, y \in \mathrm{GF}(q^n)$ *we have that* $\phi(x + y) = \phi(x) + \phi(y)$.
- ϕ *is multiplicative, i.e.,* $\forall x, y \in \mathrm{GF}(q^n)$ *we have that* $\phi(xy) = \phi(x) \cdot \phi(y)$.

This homomorphism also induces a homomorphism $\mathrm{M}_k(\mathrm{GF}(q^n)) \hookrightarrow \mathrm{M}_{nk}(\mathrm{GF}(q))$.

This lemma is a standard tool in algebra, but for completeness we give the proof.

Proof. $\mathrm{GF}(q^n)$ is an n dimensional vector space over $\mathrm{GF}(q)$. Each element $x \in \mathrm{GF}(q^n)$ can be viewed as a linear transformation $x : \mathrm{GF}(q^n) \mapsto \mathrm{GF}(q^n)$ in the following way:

$$\forall y \in \mathrm{GF}(q^n), \quad x(y) = x \cdot y.$$

Clearly this is a linear transformation of $\mathrm{GF}(q^n)$ into itself, as a vector space over $\mathrm{GF}(q)$. Therefore, by picking a basis to $\mathrm{GF}(q^n)$ we can represent the linear transformation corresponding to each $x \in \mathrm{GF}(q^n)$ by a matrix $a_x \in \mathrm{M}_n(\mathrm{GF}(q))$. Thus, we have defined a mapping $\phi : \mathrm{GF}(q^n) \mapsto \mathrm{M}_n(\mathrm{GF}(q))$ such that $\phi(x) = a_x$, and it is easy to verify that this mapping is an embedding of $\mathrm{GF}(q^n)$ into $\mathrm{M}_n(\mathrm{GF}(q))$. The way to generalize it to an embedding of $\mathrm{M}_k(\mathrm{GF}(q^n))$ into $\mathrm{M}_{nk}(\mathrm{GF}(q))$ is the following. Let $a = (a_{i,j}) \in \mathrm{M}_k(\mathrm{GF}(q^n))$ be some matrix. Every entry of $a_{i,j}$ of a is some element of $\mathrm{GF}(q^n)$. We can now replace $a_{i,j}$ with the matrix $\phi(a_{i,j})$. Thus the resulting matrix will be a $kn \times kn$ matrix whose entries are in $\mathrm{GF}(q)$. Again it is easy to verify that this is indeed an embedding of $\mathrm{M}_k(\mathrm{GF}(q^n))$ into $\mathrm{M}_{nk}(\mathrm{GF}(q))$. □

In addition to the algebraic lemmas we also need the following combinatorial tools.

DEFINITION 3.5. *Let* F *be a field, and let* v, u *be two vectors in* F^m. *We denote by* $\mathbf{weight}(v)$ *the number of nonzero coordinates of* v. *Let* $\mathbf{d_H}(v, u) = \mathbf{weight}(v - u)$; *i.e.,* $\mathbf{d_H}(v, u)$ *is the number of coordinates on which* u *and* v *differ.* $\mathbf{d_H}(v, u)$ *is also known as the Hamming distance of* u *and* v. *We also denote by* $\mathbf{agree}(u, v)$ *the number of coordinates on which* u *and* v *are equal; i.e.,* $\mathbf{agree}(u, v) = m - \mathbf{d_H}(v, u)$.

The next lemma shows that if a vector space contains a set of vectors such that every pair/triplet of them disagrees on many coordinates (i.e., their Hamming distance

is large), then it is of large dimension. There are numerous similar lemmas in coding theory, and in particular the first part of our lemma is the famous Plotkin bound (see [Lin92]).

LEMMA 3.6.

1. *In every set of k vectors in $\mathrm{GF}(q)^t$ such that $q < k$ there are two vectors that agree on at least $(\frac{t}{q} - \frac{t}{k})$ coordinates.*
2. *In every set of k vectors in $\mathrm{GF}(q)^t$ such that $2q < k$ there are three vectors that agree on at least $(\frac{t}{q^2} - \frac{3t}{qk})$ coordinates.*

Proof. We begin by proving the first claim. Let $v_1, \ldots, v_k$ be k vectors in $\mathrm{GF}(q)^t$. We are going to estimate $\sum_{i<j} \mathbf{agree}(v_i, v_j)$ in two different ways. On the one hand, this sum is at most $\binom{k}{2}$ times the maximum of $\mathbf{agree}(v_i, v_j)$. On the other hand, consider a certain coordinate. For every $\alpha \in \mathrm{GF}(q)$ denote by n_α the number of vectors among the v_i's that are equal to α on this coordinate. Clearly $\sum_{\alpha=0}^{q-1} n_\alpha = k$. The contribution of this coordinate to $\sum_{i<j} \mathbf{agree}(v_i, v_j)$ is exactly $\sum_{\alpha=0}^{q-1} \binom{n_\alpha}{2}$. By convexity

$$\sum_{\alpha=0}^{q-1} \binom{n_\alpha}{2} \geq p \cdot \frac{1}{2} \cdot \frac{k}{q} \left(\frac{k}{q} - 1 \right) = \frac{k(k-q)}{2q} .$$

We get that

$$\binom{k}{2} \cdot \max_{i<j}(\mathbf{agree}(v_i, v_j)) \geq \sum_{i<j} \mathbf{agree}(v_i, v_j) \geq t \cdot \frac{k(k-q)}{2q} .$$

Therefore

$$\max_{i<j}(\mathbf{agree}(v_i, v_j)) \geq \frac{t}{q} \cdot \frac{k-q}{k-1} \geq \frac{t}{q} \cdot \frac{k-q}{k} = \frac{t}{q} - \frac{t}{k} .$$

The proof of the second claim is similar. We give two different estimates for our $\sum_{i<j<l} \mathbf{agree}(v_i, v_j, v_l)$ (the number of coordinates on which v_i, v_j, and v_l are the same). In the same manner as before we get that

$$\max_{i<j<l}(\mathbf{agree}(v_i, v_j, v_l)) \geq \frac{t}{q^2} \cdot \frac{k-q}{k-1} \cdot \frac{k-2q}{k-2} \geq \frac{t}{q^2} - \frac{3t}{qk} . \qquad \square$$

COROLLARY 3.7. *If $\mathrm{GF}(2)$ contains k vectors $v_1, \ldots, v_k$ such that $2 < k$ and $\mathbf{d_H}(v_i, v_j) \geq N \ \forall i \neq j$, then $t \geq 2N - 4\frac{N}{k+2}$.*

Proof. According to Lemma 3.6 there are two vectors, w.l.o.g. v_1 and v_2, such that $\mathbf{agree}(v_1, v_2) \geq \frac{t}{2} - \frac{t}{k}$. Since $\mathbf{d_H}(v_1, v_2) = t - \mathbf{agree}(v_1, v_2)$, we get that

$$t - \left(\frac{t}{2} - \frac{t}{k} \right) \geq \mathbf{d_H}(v_1, v_2) \geq N,$$

and the result follows. $\square$

4. Lower bound over GF(2). In this section we prove our main theorems.

THEOREM 4.1. $\mathrm{bl}_*^{\mathrm{GF}(2)}(\mathrm{MP}_n) \geq 3n^2 - O(n^{\frac{5}{3}})$ *(in other words, $\mathrm{R}_{\mathrm{GF}(2)}(\mathrm{MP}_n) \geq 3n^2 - O(n^{\frac{5}{3}})$).*

The second theorem that we shall prove is a lower bound for quadratic circuits.

THEOREM 4.2. $q_*^{GF(2)}(MP_n) \geq 3n^2 - O(n^{\frac{5}{3}})$; *i.e., the number of product gates in any quadratic circuit that computes the product of two* $n \times n$ *matrices over* GF(2) *is at least* $3n^2 - O(n^{\frac{5}{3}})$.

Clearly Theorem 4.2 implies Theorem 4.1, but we first prove Theorem 4.1, as its proof is more intuitive and simple. We begin by introducing the notion of linear codes of matrices.

4.1. Linear codes of matrices.

DEFINITION 4.3. *A linear code of matrices is a mapping*

$$\Gamma : M_n(GF(2)) \mapsto GF(2)^m$$

(for some m*) with the following properties:*

- Γ *is linear.*
- *For any matrix* a, $\mathbf{weight}(\Gamma(a)) \geq n \cdot \mathrm{rank}(a)$.

From the linearity of Γ and the requirement on $\mathbf{weight}(\Gamma(a))$ we get the following corollary.

COROLLARY 4.4. Γ *is a one-to-one mapping, and for any two matrices* a *and* b, $\mathbf{d_H}(\Gamma(\mathrm{a}), \Gamma(\mathrm{b})) \geq n \cdot \mathrm{rank}(\mathrm{a} - \mathrm{b})$.

The following theorem shows that the dimension of the range of any linear code of matrices is large (i.e., m must be large).

THEOREM 4.5. *Let* $\Gamma : M_n(GF(2)) \mapsto GF(2)^m$ *be a linear code of matrices; then* $m \geq 3n^2 - O(n^{\frac{5}{3}})$.

Proof. Define

$$\Gamma(a) = (\mu_1(a), \ldots, \mu_m(a)).$$

The proof is based on the following lemma that shows that we can find $k = n^{\frac{1}{3}}$ matrices $a_1, \ldots, a_k \in M_n(GF(2))$, with the following properties:

- $\forall i \neq j$, $a_i - a_j$ is an invertible matrix.
- There are $n^2 - \binom{k}{2}n$ linear forms among the μ_i's that vanish on all the a_i's.

We state the lemma for every $k < 2^n$, but we apply it only to $k = n^{\frac{1}{3}}$.

LEMMA 4.6. *For every* n, k *such that* $k < 2^n$, *and for any* $\mu_1, \ldots, \mu_{n^2}$ *linearly independent linear forms in* n^2 *variables, over* GF(2), *there are* k *matrices* $a_1, \ldots, a_k \in M_n(GF(2))$ *such that for every* $i \neq j$, $a_i - a_j$ *is an invertible matrix, and such that* $n^2 - \binom{k}{2}n$ *of the* μ_i*'s vanish on them.*

Proof. Consider the following polynomial P in k matrices:

$$P(a_1, \ldots, a_k) = \text{determinant} \left(\prod_{i<j} (a_i - a_j) \right).$$

Clearly a set of k matrices $a_1, \ldots, a_k$ satisfy $P(a_1, \ldots, a_k) \neq 0$ iff all the matrices $a_i - a_j$ are invertible. In addition, it is easy to see that $\deg(P) = \binom{k}{2}n$. Therefore if we show that $P \not\equiv 0$ over GF(2), then according to Lemma 3.3 we will get what we wanted to prove.

In order to show that $P \not\equiv 0$ we just have to prove the existence of k matrices such that the difference of every two of them is invertible. Lemma 3.4 assures us that we can embed the field $GF(2^n)$ into $M_n(GF(2))$. Denote this embedding by $\Phi : GF(2^n) \hookrightarrow M_n(GF(2))$. We take k distinct elements in $GF(2^n)$, $x_1, \ldots, x_k$. Their images, $\Phi(x_1), \ldots, \Phi(x_k)$, are matrices in $M_n(GF(2))$ such that the difference of every

two of them, $\Phi(x_i) - \Phi(x_j) = \Phi(x_i - x_j)$, is an invertible matrix. This is because the x_i's are distinct (i.e., $x_i - x_j \neq 0$), and every nonzero element in $\mathrm{GF}(2^n)$ is invertible. Thus, $\Phi(x_1), \ldots, \Phi(x_k)$ are exactly the k matrices that we were looking for. This concludes the proof of the lemma. □

We proceed with the proof of Theorem 4.5. Let $k = n^{\frac{1}{3}}$. Since Γ is a one-to-one mapping, there are n^2 independent linear forms among $\mu_1, \ldots, \mu_m$. Therefore we can use Lemma 4.6 and get that there are k matrices $a_1, \ldots, a_k$ such that for every $i \neq j$, $a_i - a_j$ is invertible, and such that, w.l.o.g., $\mu_{m-r+1}, \ldots, \mu_m$ vanish on $a_1, \ldots, a_k$ for some $r \geq n^2 - \binom{k}{2} n \geq n^2 - n^{\frac{5}{3}}$.

Since the last r linear forms vanish on all the a_i's, we are going to restrict our attention to only the first $m - r$ linear forms. Thus from now on we consider only $\Gamma(a_i)$ restricted to its first $m - r$ coordinates.

Since each of the differences $a_i - a_j$ $(\forall i \neq j)$ is an invertible matrix, we get that $\mathbf{d_H}(\Gamma(a_i), \Gamma(a_j)) \geq n^2$. Thus, $\Gamma(a_1), \ldots, \Gamma(a_k)$ are k vectors contained in $\mathrm{GF}(2)^{m-r}$ (we consider only their first $m - r$ coordinates!) such that the Hamming distance of every pair of them is at least n^2. Therefore according to Corollary 3.7 we get that

$$m - r \geq 2n^2 - 4\frac{n^2}{k+2} .$$

Since $r \geq n^2 - n^{\frac{5}{3}}$ and $k = n^{\frac{1}{3}}$, we get that

$$m \geq 3n^2 - O(n^{\frac{5}{3}}),$$

which is what we wanted to prove. This concludes the proof of the theorem. □

4.2. Proof of Theorem 4.1. Assume that $\mathrm{bl}_*^{\mathrm{GF}(2)}(\mathrm{MP}_n) = m$. Let C be a smallest bilinear circuit for MP_n. Let

$$\mu_1(x) \cdot \eta_1(y), \ \ldots, \ \mu_m(x) \cdot \eta_m(y)$$

be the m bilinear forms computed in the product gates of C. We will show that these bilinear forms define in a very natural way a code on $\mathrm{M}_n(\mathrm{GF}(2))$. The code thus defined will have the property that the dimension of the space, into which the code maps $\mathrm{M}_n(\mathrm{GF}(2))$, is exactly m. Thus, according to Theorem 4.5 we will get that $m \geq 3n^2 - O(n^{\frac{5}{3}})$, which is what we wanted to prove.

We begin by defining a mapping from $\mathrm{M}_n(\mathrm{GF}(2))$ to $\mathrm{GF}(2)^m$. Let $\Gamma : \mathrm{M}_n(\mathrm{GF}(2)) \mapsto \mathrm{GF}(2)^m$ be the following mapping:

$$\Gamma(x) = (\mu_1(x), \ldots, \mu_m(x)).$$

Notice that we ignore the η_i's in the definition of Γ.

LEMMA 4.7. *Γ is a linear code of matrices.*

Proof. Clearly Γ is a linear transformation from $\mathrm{M}_n(\mathrm{GF}(2))$ to $\mathrm{GF}(2)^m$. Thus we only have to prove that, for every matrix $x \in \mathrm{M}_n(\mathrm{GF}(2))$, $\mathbf{weight}(\Gamma(x)) \geq n \cdot \mathrm{rank}(x)$. Let x be a matrix of rank r. Assume w.l.o.g. that $\mu_1(x) = \cdots = \mu_k(x) = 1$ and that $\mu_{k+1}(x) = \cdots = \mu_m(x) = 0$, i.e., $\mathbf{weight}(\Gamma(x)) = k$. We shall show that $k \geq nr$. For every $y \in \mathrm{M}_n(\mathrm{GF}(2))$, the n^2 entries of $x \cdot y$ are determined by the values of $\mu_1(x) \cdot \eta_1(y), \ldots, \mu_m(x) \cdot \eta_m(y)$. Since $\mu_{k+1}(x) = \cdots = \mu_m(x) = 0$, we get that $\eta_1(y), \ldots, \eta_k(y)$ determine $x \cdot y$. Therefore there are at most 2^k different matrices of the form $x \cdot y$. Since $\mathrm{rank}(x) = r$, we get that there are exactly 2^{nr} different matrices of the form $x \cdot y$. Therefore $k \geq nr$. This concludes the proof of the lemma. □

Therefore Γ is a linear code of matrices, and so according to Theorem 4.5 we get that $m \geq 3n^2 - O(n^{\frac{5}{3}})$, which is what we wanted to prove. This concludes the proof of Theorem 4.1. □

4.3. Proof of Theorem 4.2. As in the proof of Theorem 4.1, we will show that every quadratic circuit for MP_n defines a code on $\mathrm{M}_n(\mathrm{GF}(2))$. The code thus defined will have the property that m (i.e., the dimension of the space into which the code maps $\mathrm{M}_n(\mathrm{GF}(2))$) is exactly the number of product gates in the circuit. Thus, according to Theorem 4.5 we will get that $m \geq 3n^2 - O(n^{\frac{5}{3}})$, which is what we wanted to prove.

Let C be a quadratic circuit for MP_n. Assume that the product gates of C compute the quadratic forms $\mu_1(x,y) \cdot \eta_1(x,y), \ldots, \mu_m(x,y) \cdot \eta_m(x,y)$. Thus, each of the outputs $(x \cdot y)_{i,j}$ can be written as a sum of these quadratic forms:

$$(x \cdot y)_{i,j} = \sum_{k=1}^{m} \alpha_{i,j}^{(k)} \cdot \mu_k(x,y) \cdot \eta_k(x,y),$$

where $\alpha_{i,j}^{(k)} \in \mathrm{GF}(2)$.

We would like to have a proof similar to the proof of Theorem 4.1. In that proof we defined a code of matrices using the linear transformation $\mu_1, \ldots, \mu_m$. Unfortunately that method will fail here, as μ_i is a linear function in both the variables of x and the variables of y and not just in the variables of x as in the proof of Theorem 4.1. In order to overcome this obstacle we introduce a new set of variables $z = \{z_{i,j}\}_{i,j=1,\ldots,n}$. We think about z as an $n \times n$ matrix. Define the following m linear forms in z:

$$\gamma_k(z) = \sum_{i,j} \alpha_{i,j}^{(k)} z_{i,j}, \quad k = 1, \ldots, m.$$

We get that

$$\begin{aligned}
&\sum_{k=1}^{m} \mu_k(x,y) \cdot \eta_k(x,y) \cdot \gamma_k(z) \\
&= \sum_{k=1}^{m} \left(\sum_{i,j} z_{i,j} \cdot \alpha_{i,j}^{(k)} \right) \cdot \mu_k(x,y) \cdot \eta_k(x,y) \\
&= \sum_{i,j} z_{i,j} \sum_{k=1}^{m} \alpha_{i,j}^{(k)} \cdot \mu_k(x,y) \cdot \eta_k(x,y) \qquad (4.1) \\
&= \sum_{i,j} z_{i,j} \cdot (x \cdot y)_{i,j} = \mathbf{trace}(x \cdot y \cdot z^t),
\end{aligned}$$

where $(z^t)_{i,j} = z_{j,i}$. The computation just performed shows that the γ_k's that we introduced are quite natural. We also notice that z plays the same role in $\mathbf{trace}(x \cdot y \cdot z^t)$ as do x and y. These observations motivate us to try to repeat the proof of Theorem 4.1 using the γ_k's instead of the μ_i's.

Define a linear mapping $\Gamma : \mathrm{M}_n(\mathrm{GF}(2)) \mapsto \mathrm{GF}(2)^m$ by

$$\Gamma(z) = (\gamma_1(z), \ldots, \gamma_m(z)).$$

LEMMA 4.8. *Γ is a linear code of matrices.*

Proof. Clearly Γ is a linear mapping. So we have to prove only the claim about the weights. Let z_0 be a matrix of rank r, and assume w.l.o.g. that $\gamma_1(z_0) = \cdots = \gamma_k(z_0) = 1$ and $\gamma_{k+1}(z_0) = \cdots = \gamma_m(z_0) = 0$. We wish to prove that $k \geq nr$. From (4.1) we get that

$$\mathbf{trace}(x \cdot y \cdot {z_0}^t) = \sum_{h=1}^{k} \mu_h(x,y) \cdot \eta_h(x,y).$$

We now consider the discrete derivatives of this equation. Let $e_{i,j}$ be the matrix of all zeros but one in the (i,j)th place. Define

$$\frac{\partial}{\partial x_{i,j}} \mathbf{trace}(x \cdot y \cdot {z_0}^t) \stackrel{\text{def}}{=} \mathbf{trace}((x + e_{i,j}) \cdot y \cdot {z_0}^t) - \mathbf{trace}(x \cdot y \cdot {z_0}^t).$$

On the one hand,

$$\mathbf{trace}((x + e_{i,j}) \cdot y \cdot {z_0}^t) - \mathbf{trace}(x \cdot y \cdot {z_0}^t) = \mathbf{trace}(e_{i,j} \cdot y \cdot {z_0}^t) = (z_0 \cdot y^t)_{i,j}.$$

On the other hand, we have that

$$\mathbf{trace}((x + e_{i,j}) \cdot y \cdot {z_0}^t) - \mathbf{trace}(x \cdot y \cdot {z_0}^t)$$

$$= \sum_{h=1}^{k} (\mu_h(x + e_{i,j}, y) \cdot \eta_h(x + e_{i,j}, y) - \mu_h(x,y) \cdot \eta_h(x,y))$$

$$= \sum_{h=1}^{k} (\mu_h(e_{i,j}, 0) \cdot \eta_h(x,y) + \mu_h(x,y) \cdot \eta_h(e_{i,j}, 0)) \tag{4.2}$$

$$+ \sum_{h=1}^{k} \mu_h(e_{i,j}, 0) \cdot \eta_h(e_{i,j}, 0),$$

where the last equality follows from the linearity of the μ_h's and the η_h's. Since $(z_0 \cdot y^t)_{i,j}$ is a linear form in y, we actually get that

$$(z_0 \cdot y^t)_{i,j} = \frac{\partial}{\partial x_{i,j}} \mathbf{trace}(x \cdot y \cdot {z_0}^t)$$

$$= \sum_{h=1}^{k} (\mu_h(e_{i,j}, 0) \cdot \eta_h(x,y) + \mu_h(x,y) \cdot \eta_h(e_{i,j}, 0))$$

$$\in \mathbf{span}(\mu_h(x,y),\ \eta_h(x,y))$$

in the vector space of all linear forms in x, y. (Since $(z_0 \cdot y^t)_{i,j}$ is a linear form, the third summand of (4.2) sums to 0.) In the same manner we define

$$\frac{\partial}{\partial y_{i,j}}\mathbf{trace}(x \cdot y \cdot {z_0}^t) \stackrel{\text{def}}{=} \mathbf{trace}(x \cdot (y + e_{i,j}) \cdot {z_0}^t) - \mathbf{trace}(x \cdot y \cdot {z_0}^t).$$

We get that

$$(x^t \cdot z_0)_{i,j} = \frac{\partial}{\partial y_{i,j}}\mathbf{trace}(x \cdot y \cdot {z_0}^t)$$

$$= \sum_{h=1}^{k}(\mu_h(0, e_{i,j}) \cdot \eta_h(x, y) + \mu_h(x, y) \cdot \eta_h(0, e_{i,j}))$$

$$\in \mathbf{span}(\mu_h(x, y),\ \eta_h(x, y)).$$

Denote by PD the set of all the discrete partial derivatives

$$\left\{\frac{\partial}{\partial x_{i,j}}\mathbf{trace}(x \cdot y \cdot {z_0}^t), \frac{\partial}{\partial y_{i,j}}\mathbf{trace}(x \cdot y \cdot {z_0}^t)\right\}_{i,j}.$$

We just proved that PD is contained in the linear span of

$$\{\mu_h(x, y),\ \eta_h(x, y)\}_{h=1}^{k}.$$

Therefore

$$\dim(\mathbf{span}(\mathrm{PD})) \le \dim(\mathbf{span}\{\mu_h(x, y),\ \eta_h(x, y)\}_{h=1}^{k}) \le 2k. \tag{4.3}$$

We also showed that

$$\left\{\frac{\partial}{\partial x_{i,j}}\mathbf{trace}(x \cdot y \cdot {z_0}^t), \frac{\partial}{\partial y_{i,j}}\mathbf{trace}(x \cdot y \cdot {z_0}^t)\right\}_{i,j}$$

$$= \left\{(x^t \cdot z_0)_{i,j},\ (z_0 \cdot y^t)_{i,j}\right\}_{i,j}.$$

Therefore, using our assumption that $\mathrm{rank}(z_0) = r$, we get that

$$\dim(\mathbf{span}(\mathrm{PD})) = \dim(\mathbf{span}\left\{(x^t \cdot z_0)_{i,j},\ (z_0 \cdot y^t)_{i,j}\right\}_{i,j}) = 2nr. \tag{4.4}$$

Combining (4.3) and (4.4), we get that $2k \ge 2nr$. □

Theorem 4.2 now follows from applying Theorem 4.5 to the linear code of matrices Γ. □

5. Other finite fields. In this section we prove the following theorem.

THEOREM 5.1. *The number of product gates in any bilinear circuit that computes the product of two $n \times n$ matrices over* $\mathrm{GF}(q)$ *is at least* $(2.5 + \frac{1.5}{q^3-1})n^2 - O(n^{\frac{7}{4}})$ *(i.e.,* $\mathrm{bl}_*^{\mathrm{GF}(q)}(\mathrm{MP}_n) \ge (2.5 + \frac{1.5}{q^3-1})n^2 - O(n^{\frac{7}{4}})$ *over* $\mathrm{GF}(q)$*).*

Let C be a bilinear circuit for MP_n over $\mathrm{GF}(q)$. Assume that $\mu_1(x) \cdot \eta_1(y), \ldots, \mu_m(x) \cdot \eta_m(y)$ are the bilinear forms computed in the product gates of C. The following lemma of Bläser is the main tool in the proof of the theorem.

LEMMA 5.2 (see [Bla99]). *Let $[a,b] = ab - ba$. If there are two matrices a, b such that $[a,b]$ is an invertible matrix and such that there are t linear forms among $\mu_1, \ldots, \mu_m$ such that each of them vanishes on I, a, b, then $m \geq t + 1.5n^2$.*

We are going to prove that we can find a, b such that $(1 - \frac{1}{q^3})n^2 + \frac{m}{q^3} - O(n^{\frac{7}{4}})$ linear forms among $\mu_1, \ldots, \mu_m$ vanish on I, a, b and such that $[a,b]$ is invertible.

We are going to prove the theorem only for n even. (This certainly implies the lower bound for odd n as well.) Thus from now on we assume that *n is an even number*.

Proof of Theorem 5.1. We begin by proving that (w.l.o.g.) many of the μ_i's vanish on I. The following lemma shows that we can always find an invertible matrix such that many of the μ_i's vanish on it. As before, we assume that $\mu_1, \ldots, \mu_{n^2}$ are independent linear forms.

LEMMA 5.3. *There exists an invertible matrix c such that at least $(1 - \frac{1}{q})n^2 + \frac{m}{q} - O(n^{\frac{5}{3}})$ of the μ_i's vanish on it, where $n^2 - O(n^{\frac{5}{3}})$ of the μ_i's that vanish on it are among $\mu_1, \ldots, \mu_{n^2}$.*

In the proof of the lemma we assume for simplicity that $n^2 < m < 10n^2$, as it will not change the results.

Proof. An analogue of Lemma 4.6 over $\mathrm{GF}(q)$ guarantees that we can find $k = n^{\frac{1}{3}}$ matrices $a_1, \ldots, a_k \in \mathrm{M}_n(\mathrm{GF}(q))$ such that for $i \neq j$, $a_i - a_j$ is invertible and such that $n^2 - \binom{k}{2}n$ linear forms among $\mu_1, \ldots, \mu_{n^2}$ vanish on all of the a_i's. Define $r = n^2 - \binom{k}{2}n$, and assume w.l.o.g. that $\mu_1, \ldots, \mu_r$ vanish on all the a_i's.

Let us consider the following k vectors in $\mathrm{GF}(q)^{m-r}$:

$$\Gamma(a_i) \stackrel{\mathrm{def}}{=} (\mu_{r+1}(a_i), \ldots, \mu_m(a_i)), \quad i = 1, \ldots, k\ .$$

As in the proof of Theorem 4.1, we get that since $a_i - a_j$ is an invertible matrix $\forall i \neq j$, then $\mathbf{d_H}(\Gamma(a_i), \Gamma(a_j)) \geq n^2$. According to Lemma 3.6, two of these vectors agree on at least $\frac{m-r}{q} - \frac{m-r}{k}$ coordinates. Assume that $\Gamma(a_1)$ and $\Gamma(a_2)$ are these vectors. Define $c = a_1 - a_2$. We have that c is an invertible matrix such that the first $r = n^2 - \binom{k}{2}n$ linear forms (which are independent) vanish on it and such that all the linear forms that $\Gamma(a_1)$ and $\Gamma(a_2)$ agree on vanish on it as well. Therefore there are at least

$$r + \frac{m-r}{q} - \frac{m-r}{k}$$

linear forms that vanish on c. Since $r = n^2 - \binom{k}{2}n$ and $k = n^{\frac{1}{3}}$, we get that at least

$$\left(1 - \frac{1}{q}\right)n^2 + \frac{m}{q} - O(n^{\frac{5}{3}})$$

linear forms vanish on c, and $n^2 - O(n^{\frac{5}{3}})$ of them are among $\mu_1, \ldots, \mu_{n^2}$. This completes the proof of the lemma. ∎

Lemma 5.3 does not specify c, but using the *sandwiching method* (see below), we can assume that $c = I$: We know that $x \cdot y$ is computed using the bilinear forms

$$\mu_1(x) \cdot \eta_1(y), \ldots, \mu_m(x) \cdot \eta_m(y).$$

We now do the following trick: $x \cdot y = (x \cdot c) \cdot (c^{-1} \cdot y)$; therefore $x \cdot y$ can be computed using the bilinear forms

$$\tilde{\mu_1}(x) \cdot \tilde{\eta_1}(y), \ldots, \tilde{\mu_m}(x) \cdot \tilde{\eta_m}(y),$$

where

$$\tilde{\mu_i}(x) \stackrel{\text{def}}{=} \mu_i(x \cdot c) \quad \text{and} \quad \tilde{\eta_i}(y) \stackrel{\text{def}}{=} \eta_i(c^{-1} \cdot y).$$

Thus, if $\mu_i(c) = 0$, then we get that $\tilde{\mu_i}(I) = \mu_i(I \cdot c) = 0$. This trick is called sandwiching; for further background, see [Bla99, Gro78].

Thus, by combining the sandwiching method and Lemma 5.3, we get that we can assume w.l.o.g. that $(1 - \frac{1}{q})n^2 + \frac{m}{q} - O(n^{\frac{5}{3}})$ of the μ_i's, where $n^2 - O(n^{\frac{5}{3}})$ of them are among $\mu_1, \ldots, \mu_{n^2}$, vanish on I. The next lemma now assures us that we can find two matrices a, b that satisfy the requirements of Lemma 5.2.

LEMMA 5.4. *There are two matrices a, b such that $[a, b]$ is an invertible matrix and such that at least $(1 - \frac{1}{q^3})n^2 + \frac{m}{q^3} - O(n^{\frac{7}{4}})$ of the μ_i's vanish on I, a, b.*

Proof. The proof of this lemma is similar to the proof of Lemma 5.3. Let $k = n^{\frac{1}{4}}$. The following lemma shows that we can find k matrices such that many of the μ_i's vanish on all of them and such that among their differences there are matrices satisfying the requirements of Lemma 5.2.

LEMMA 5.5. *For every n, k such that $q^{\frac{n}{2}} > 4\binom{k}{3}$, and for any $\mu_1, \ldots, \mu_{n^2}$ linearly independent linear forms, in n^2 variables, over $\mathrm{GF}(q)$, there are k matrices $a_1, \ldots, a_k$ such that $\forall i < j < l$, $[a_i - a_l, a_j - a_l]$ is invertible and such that $n^2 - 2\binom{k}{3}n$ of the μ_i's vanish on all the a_i's.*

Proof. Again we use Lemma 3.3. Let P be the following polynomial:

$$P(a_1, \ldots, a_k) = \text{determinant}\left(\prod_{i<j<l} [a_i - a_l, a_j - a_l]\right).$$

Clearly $\deg(P) = 2\binom{k}{3}n$ (as a polynomial in the entries of the a_i's). Therefore if we prove that $P \not\equiv 0$, i.e., that there exist k matrices on which P is not zero, then according to Lemma 3.3 we are done. This is guaranteed by the following lemma.

LEMMA 5.6. *If $q^{\frac{n}{2}} > 4\binom{k}{3}$, then there exist k matrices in $\mathrm{M}_n(\mathrm{GF}(q))$, $a_1, \ldots, a_k$, such that $\forall i < j < l$, $[a_i - a_l, a_j - a_l]$ is invertible.*

This lemma is the reason that we assume that n is even. We do not know how to prove it for odd values of n.

Proof. Consider the following polynomial in $\mathbf{4k}$ variables (i.e., a polynomial in k matrices over $\mathrm{M}_{\mathbf{2}}(\mathrm{GF}(\mathbf{q}^{\frac{\mathbf{n}}{\mathbf{2}}}))$!):

$$Q(x_1, \ldots, x_k) = \text{determinant}\left(\prod_{i<j<l} [x_i - x_l, x_j - x_l]\right).$$

Q is a polynomial of degree $d = 4\binom{k}{3}$ over $\mathrm{GF}(q)$ in the entries of the a_i's. Clearly Q is not the zero polynomial (as it is a product of nonzero polynomials). Consider the field $F = \mathrm{GF}(q^{\frac{n}{2}})$. Since $d < |\mathrm{F}|$, we get by Lemma 3.1 that there are k matrices $\rho_1, \ldots, \rho_k \in \mathrm{M}_2(F)$ such that $Q(\rho_1, \ldots, \rho_k) \neq 0$. That is, $\forall i < j < l$, $[\rho_i - \rho_l, \rho_j - \rho_l]$ is an invertible matrix. According to Lemma 3.4 we can embed $\mathrm{M}_2(F)$ in $\mathrm{M}_n(\mathrm{GF}(q))$. Therefore there are k matrices in $\mathrm{M}_n(\mathrm{GF}(q))$ satisfying $\forall i < j < l$, $[a_i - a_l, a_j - a_l]$ is an invertible matrix, which is what we wanted to prove. □

This concludes the proof of Lemma 5.5. □

We proceed with the proof of Lemma 5.4. We now restrict our attention to the linear forms among $\mu_{n^2+1}, \ldots, \mu_m$ that vanish on I. We shall prove that three of the matrices guaranteed by Lemma 5.5 agree on many of these linear forms (more

formally on $\frac{m-n^2}{q^3} - O(n^{\frac{7}{4}})$ of them). Thus, if a_1, a_2, a_3 are these three matrices, then we get that $(1-\frac{1}{q^3})n^2 + \frac{m}{q^3} - O(n^{\frac{7}{4}})$ linear forms vanish on I, $(a_1 - a_3)$, $(a_2 - a_3)$, and that $[(a_1 - a_3), (a_2 - a_3)]$ is an invertible matrix, which is what we wanted to prove.

Assume w.l.o.g. that the linear forms $\mu_{n^2+1}, \ldots, \mu_{n^2+r}$ vanish on I (beside those among $\mu_1, \ldots, \mu_{n^2}$ that vanish on it), where $r \geq \frac{m-n^2}{q} - O(n^{\frac{5}{3}})$. Let $a_1, \ldots, a_k$ be the matrices guaranteed by Lemma 5.5. Consider the following vectors: $\forall 1 \leq i \leq k$,

$$v_i = (\mu_{n^2+1}(a_i), \ldots, \mu_{n^2+r}(a_i)) \in \mathrm{GF}(q)^r \ .$$

According to Lemma 3.6 three of these vectors, namely v_1, v_2, v_3, agree on at least $\frac{r}{q^2} - \frac{3r}{qk}$ coordinates. Therefore there are $\frac{r}{q^2} - \frac{3r}{qk}$ linear forms among $\mu_{n^2+1}, \ldots, \mu_{n^2+r}$ that vanish on $a_1 - a_3$ and $a_2 - a_3$. In addition there are $n^2 - 2\binom{k}{3}n$ linear forms among $\mu_1, \ldots, \mu_{n^2}$ that vanish on a_1, a_2, a_3; hence there are $n^2 - 2\binom{k}{3}n$ linear forms among $\mu_1, \ldots, \mu_{n^2}$ that vanish on $a_1 - a_3$ and on $a_2 - a_3$. Let $a = a_1 - a_3$, $b = a_2 - a_3$.

We get that there are $\frac{r}{q^2} - \frac{3r}{qk}$ linear forms among $\mu_{n^2+1}, \ldots, \mu_{n^2+r}$ that vanish on I, a, b. Since $n^2 - O(n^{\frac{5}{3}})$ of the first n^2 μ_i's vanish on I, we get that at least $n^2 - 2\binom{k}{3}n - O(n^{\frac{5}{3}})$ of the first n^2 μ_i's vanish on I, a, b. Putting it all together, we get that at least

$$n^2 - 2\binom{k}{3}n - O(n^{\frac{5}{3}}) + \frac{r}{q^2} - \frac{3r}{qk}$$

linear forms among $\mu_1, \ldots, \mu_m$ vanish on I, a, b. Since $r \geq \frac{m-n^2}{q} - O(n^{\frac{5}{3}})$ and $k = n^{\frac{1}{4}}$, we get that at least

$$\left(1 - \frac{1}{q^3}\right) n^2 + \frac{m}{q^3} - O(n^{\frac{7}{4}})$$

of the μ_i's vanish on I, a, b as well. This concludes the proof of Lemma 5.4. □

Putting everything together, we get by Lemmas 5.2 and 5.4 that

$$m \geq 1.5n^2 + \left(1 - \frac{1}{q^3}\right) n^2 + \frac{m}{q^3} - O(n^{\frac{7}{4}}).$$

Therefore

$$m \geq \left(2.5 + \frac{1.5}{q^3 - 1}\right) n^2 - O(n^{\frac{7}{4}}).$$

This concludes the proof of Theorem 5.1. □

6. Limits of our technique. The proof of our lower bound over GF(2) was based on a coding theoretic argument. We claimed that every code with a certain rate and minimal distance must have high length (Theorem 4.5). This lower bound implied the lower bound for bilinear circuits over GF(2). We also notice that any improvement of the lower bound in the theorem will improve, in the same way, our lower bound for bilinear circuits. The best techniques today are due to McEliece et al. [MRRW77], which yield a lower bound of $\approx m \geq 3.57n^2$ on the minimal m for which there is a linear code from $\mathrm{GF}(2)^{n^2}$ to $\mathrm{GF}(2)^m$, of minimal distance n^2. Since any such linear code also gives a code of matrices, we cannot hope to prove anything better than $m \geq 3.57n^2$ (unless we improve on [MRRW77], which is a major open

problem). On the other hand, we know that there are such linear codes with, say, $m < 5n^2$ (e.g., random linear codes). So even if we do improve on [MRRW77], we cannot prove any lower bound better than $5n^2$. Thus, new techniques are needed in order to prove better lower bounds.

Appendix. On the relation between $\mathbf{bl}_*^{\mathbf{F}}(\mathbf{MP}_n)$ and $\mathbf{bl}_{\mathbf{tot}}^{\mathbf{F}}(\mathbf{MP}_n)$.

THEOREM A.1. *If for some n_0 there exists a bilinear circuit that computes the product of two $n_0 \times n_0$ matrices over* F*, with ${n_0}^{2+w}$ multiplications and t additions, then for a large enough n we have that* $\mathrm{bl}_{\mathrm{tot}}^{\mathrm{F}}(\mathrm{MP}_n) = O(n^{2+w})$.

A similar claim was first proved by Strassen [Str69]. We give the proof for completeness (see also [BCS97, Proposition 15.1]).

Proof. The idea of the proof is to use Strassen's recursion method. Assume that we have a bilinear circuit that computes the product of two $n_0 \times n_0$ matrices and has m multiplication gates and t plus gates. Then, given two $n \times n$ matrices ($n \gg n_0$) X and Y, we partition each of them into ${n_0}^2$ submatrices of size $\frac{n}{n_0} \times \frac{n}{n_0}$ each. We now think about X, Y as matrices of size $n_0 \times n_0$, whose elements come from the ring of matrices. For clearance we denote by $\tilde{X}$ and $\tilde{Y}$ the matrices X and Y when we think about them as $n_0 \times n_0$ matrices whose elements are also matrices. In order to compute $X \cdot Y$ we apply the bilinear circuit for computing the product of two $n_0 \times n_0$ matrices on the input $\tilde{X}, \tilde{Y}$. When we have to compute a product of the form $\tilde{X}_{i,j} \cdot \tilde{Y}_{k,l}$, we recursively compute the product of the relevant $\frac{n}{n_o} \times \frac{n}{n_0}$ matrices. Whenever we have to compute a plus gate (which now adds two $\frac{n}{n_0} \times \frac{n}{n_0}$ matrices), we compute all the relevant $(\frac{n}{n_0})^2$ sums. We thus have the following recursion formula:

$$\mathrm{bl}_{\mathrm{tot}}^{\mathrm{F}}(\mathrm{MP}_n) \le m \cdot \mathrm{bl}_{\mathrm{tot}}^{\mathrm{F}}(\mathrm{MP}_{\frac{n}{n_0}}) + t \cdot \left(\frac{n}{n_0}\right)^2 .$$

Hence

$$\mathrm{bl}_{\mathrm{tot}}^{\mathrm{F}}(\mathrm{MP}_n) \le m^{\frac{\log n}{\log n_0}} + t \cdot \frac{n^2}{{n_0}^2} \cdot \frac{\left(\frac{m}{{n_0}^2}\right)^{\frac{\log n}{\log n_0}} - 1}{\frac{m}{{n_0}^2} - 1} .$$

Thus, if $m = (n_0)^{2+w}$ and t is arbitrary, then we get

$$\mathrm{bl}_{\mathrm{tot}}^{\mathrm{F}}(\mathrm{MP}_n) \le n^{2+w} + t \cdot \frac{n^2}{{n_o}^2} \cdot \frac{n^w - 1}{{n_0}^w - 1} = O(n^{2+w}) . \qquad \square$$

As a corollary we get that the exponents of the total complexity and of the multiplicative complexity of matrix multiplication are the same. Formally let

$$\omega^{\mathrm{tot}}(F) = \inf\{\tau \in \mathbb{R} \mid \mathrm{bl}_{\mathrm{tot}}^{\mathrm{F}}(\mathrm{MP}_n) = O(n^\tau)\}$$

and

$$\omega^*(F) = \inf\{\tau \in \mathbb{R} \mid \mathrm{bl}_*^{\mathrm{F}}(\mathrm{MP}_n) = O(n^\tau)\}.$$

Then the above theorem implies that $\omega^{\mathrm{tot}}(F) = \omega^*(F)$.

Acknowledgments. I would like to thank Michael Ben-Or and Avi Wigderson for helpful conversations, and Ran Raz for commenting on an earlier draft of the paper. I would also like to thank the anonymous referees for their comments and for bringing [Win68, Wak70] to my attention.

REFERENCES

[Bla99] M. Bläser, *A $\frac{5}{2}n^2$-lower bound for the rank of $n \times n$-matrix multiplication over arbitrary fields*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 45–50.

[Bla00] M. Bläser, *Lower bound for the multiplicative complexity of matrix multiplication*, J. Comput. Complexity, 9 (2000), pp. 73–112.

[Bla01] M. Bläser, *A $2.5n^2$-lower bound for the multiplicative complexity of $n \times n$-matrix multiplication*, in Proceedings of the 18th International GI-MIMD Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Comput. Sci. 2010, Springer-Verlag, New York, 2001, pp. 99–110.

[BD78] R. W. Brockett and D. Dobkin, *On the optimal evaluation of a set of bilinear forms*, Linear Algebra Appl., 9 (1978), pp. 207–235.

[Bsh89] N. H. Bshouty, *A lower bound for matrix multiplication*, SIAM J. Comput., 18 (1989), pp. 759–765.

[BCS97] P. Bürgisser, M. Clausen, and M. A. Shokrollahi, *Algebraic Complexity Theory*, Springer-Verlag, New York, 1997.

[CW90] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progression*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[Gat88] J. von zur Gathen, *Algebraic complexity theory*, Annu. Rev. Comput. Sci., 3 (1988), pp. 317–347.

[Gro78] H. F. de Groote, *Lectures on the Complexity of Bilinear Problems*, Lecture Notes in Comput. Sci. 245, Springer, New York, 1986.

[LW78] J. C. Lafon and S. Winograd, *A Lower Bound for the Multiplicative Complexity of the Product of Two Matrices*, Technical report, Centre de Calcul de L'Esplanade, U.E.R. de Mathematique University Louis Pasteur, Strasbourg, France, 1978.

[MRRW77] R. J. McEliece, E. R. Rodemich, H. C. Rumsey, and L. R. Welch, *New upper bounds on the rate of codes via the Delsarte–MacWilliams inequalities*, IEEE Trans. Inform. Theory, 23 (1977), pp. 157–166.

[RS01] R. Raz and A. Shpilka, *Lower bounds for matrix product, in bounded depth circuits with arbitrary gates*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, Crete, Greece, 2001, pp. 409–418.

[Sch80] J. T. Schwartz, *Fast probabilistic algorithms for verification of polynomial identities*, J. ACM, 27 (1980), pp. 701–717.

[Str69] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[Str73] V. Strassen, *Vermeidung von divisionen*, J. Reine Angew. Math., 264 (1973), pp. 182–202.

[Lin92] J. H. van Lint, *Introduction to Coding Theory*, 2nd ed., Springer-Verlag, Berlin, 1992.

[Win68] S. Winograd, *A new algorithm for inner products,* IEEE Trans. Comput., 17 (1968), pp. 693–694.

[Wak70] A. Waksman, *On Winograd's algorithm for inner products,* IEEE Trans. Comput., 19 (1970), pp. 360–361.

[Zip79] R. Zippel, *Probabilistic algorithms for sparse polynomials*, in Symbolic and Algebraic Computation (Proceedings of EUROSAM '79, International Symposium, Marseille, 1979), Springer, Berlin, 1979, pp. 216–226.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1201–1209

SEARCHING FOR SORTED SEQUENCES OF KINGS IN TOURNAMENTS*

JIAN SHEN†, LI SHENG‡, AND JIE WU§

Abstract. A *tournament* T_n is an orientation of a complete graph on n vertices. A *king* in a tournament is a vertex from which every other vertex is reachable by a path of length at most 2. A *sorted sequence of kings* in a tournament T_n is an ordered list of its vertices $u_1, u_2, \ldots, u_n$ such that u_i dominates u_{i+1} ($u_i \to u_{i+1}$) and u_i is a king in the subtournament induced by $\{u_j : i \le j \le n\}$ for each $i = 1, 2, \ldots, n-1$. In particular, if T_n is transitive, searching for a sorted sequence of kings in T_n is equivalent to sorting a set of n numbers. In this paper, we try to find a sorted sequence of kings in a general tournament by asking the following type of binary question: "What is the orientation of the edge between two specified vertices u, v?" The *cost* for finding a sorted sequence of kings is the minimum number of binary questions asked in order to guarantee the finding of a sorted sequence of kings. Using an adversary argument proposed in this paper, we show that the cost for finding a sorted sequence of kings in T_n is $\Theta(n^{3/2})$ in the worst case, thus settling the order of magnitude of this question. We also show that the cost for finding a king in T_n is $\Omega(n^{4/3})$ and $O(n^{3/2})$ in the worst case. Finally, we show a connection between a sorted sequence of kings and a median order in a tournament.

Key words. adversary argument, divide-and-conquer algorithm, king, recursive relation, sorted sequence of kings, tournament

AMS subject classifications. 05C20, 05C85, 68W40

DOI. 10.1137/S0097539702410053

1. Introduction.

A *digraph* $G = (V, E)$ contains a vertex set V and an edge set E, where each edge contains a tail u and a head v (and thus the orientation $u \to v$). For a vertex u in G, the *outdegree* $d^+(G, u)$ of u is the number of edges with tail u, and the *indegree* $d^-(G, u)$ of u is the number of edges with head u. We will use $d^+(u)$ and $d^-(u)$ to denote $d^+(G, u)$ and $d^-(G, u)$, respectively, if G is specified from the context. A *tournament* T_n is an orientation of a complete graph on n vertices; that is, between any two distinct vertices u, v, there is exactly one edge: either $u \to v$ or $v \to u$ (but not both). This concept is used to model a tournament of n players where every two players compete in a game and player u beats player v if and only if the vertex u dominates the vertex v ($u \to v$) in T_n. (Suppose no game has a draw.) A vertex u is called a *king* in T_n if every other vertex is reachable from u by a path of length at most 2; that is, for each $v \neq u$, either $u \to v$ or $u \to w \to v$ for some vertex w dependent of v. It is known [11] that a vertex with the maximum outdegree is always a king in T_n. A *sorted sequence of kings* in T_n is an ordered list of its vertices $u_1, u_2, \ldots, u_n$ such that $u_i \to u_{i+1}$ and u_i is a king in the subtournament induced by $\{u_j : i \le j \le n\}$ for each $i = 1, 2, \ldots, n-1$. In particular, if T_n is *transitive* (that is, $u \to v$ and $v \to w$ imply $u \to w$), searching for a sorted sequence of kings in T_n is

*Received by the editors June 22, 2002; accepted for publication (in revised form) May 8, 2003; published electronically August 6, 2003.
http://www.siam.org/journals/sicomp/32-5/41005.html
†Department of Mathematics, Southwest Texas State University, San Marcos, TX 78666 (js48@swt.edu).
‡Department of Mathematics, Drexel University, Philadelphia, PA 19104 (lsheng@mcs.drexel.edu). The work of this author was supported by the National Science Foundation under grant CCR-0311413 to Drexel University.
§Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 (jie@cse.fau.edu).

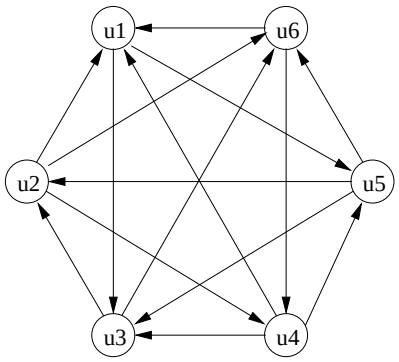


FIG. 1. *A sample tournament.*

equivalent to sorting a set of n numbers.

The existence of a sorted sequence of kings in any tournament was shown by Lou, Wu, and Sheng in [7], where an insertion sort algorithm with the worst case complexity of $O(n^3)$ was presented. A modified insertion sort algorithm with the worst case complexity of $O(n^2)$ was given by Wu and Sheng in [12]. It is easy to prove that a tournament has a unique sorted sequence of kings if and only if it is transitive. Thus in a general tournament the sorted sequence of kings is not unique. For example, Figure 1 shows the graph representation of a tournament. One sorted sequence of kings of the tournament is $u_2 \to u_4 \to u_1 \to u_5 \to u_3 \to u_6$, and another one is $u_2 \to u_6 \to u_4 \to u_1 \to u_5 \to u_3$.

In this paper, we try to find a sorted sequence of kings in T_n by asking the following type of binary question: "What is the orientation of the edge between two specified vertices u, v?" The *cost* for finding a sorted sequence of kings is the minimum number of questions asked in order to guarantee the finding of a sorted sequence of kings. In particular, if we are told that T_n is transitive, the worst case cost for finding a sorted sequence of kings in T_n is $\Theta(n \log n)$, which is the number of comparisons needed to sort n numbers in the worst case. We set to determine the worst case cost for finding a sorted sequence of kings in T_n (which may not be transitive).

For a vertex u in a digraph, let $\Gamma^+(u) = \{v : u \to v\}$ be the *first out-neighborhood* of u, $\Gamma^{++}(u) = \{w : u \to v \to w \text{ for some } v\} \setminus \Gamma^+(u)$ be the *second out-neighborhood* of u, and $\Gamma^-(u) = \{v : v \to u\}$ be the *first in-neighborhood* of u. The following lemma follows easily from the fact that each vertex in $\Gamma^+(u)$ is reachable from each vertex in $\Gamma^-(u)$ by a path of length at most 2.

LEMMA 1. *For a vertex u in T_n, let $u_1, \ldots, u_t$ be a sorted sequence of kings in the subtournament of T_n induced by $\Gamma^-(u)$, and let $u_{t+2}, \ldots, u_n$ be a sorted sequence of kings in the subtournament of T_n induced by $\Gamma^+(u)$. Then $u_1, \ldots, u_t, u, u_{t+2}, \ldots, u_n$ form a sorted sequence of kings in T_n. In particular, u_1 is a king in T_n.*

One can apply the above lemma recursively to obtain a *divide-and-conquer algorithm* for the search of a sorted sequence of kings in T_n as follows:

1. Choose a pivot vertex u arbitrarily.
2. Use $n-1$ questions to find the edge orientation between u and every other vertex, and thus obtain $\Gamma^-(u)$ and $\Gamma^+(u)$.
3. Apply the procedure recursively to $\Gamma^-(u)$ and $\Gamma^+(u)$.
4. Chain the outcomes of $\Gamma^-(u)$ and $\Gamma^+(u)$ with u in the way provided by Lemma 1.

This divide-and-conquer algorithm performs similarly to *quick sort*. It is known that quick sort has an average case performance of $\Theta(n \log n)$ and the worst case performance of $\Theta(n^2)$. The average cost for finding a sorted sequence of kings by using the above divide-and-conquer algorithm is satisfactory ($\Theta(n \log n)$) since it is equivalent to applying quick sort to n numbers [2]. On the other hand, the worst case cost by using this algorithm is $n(n-1)/2$ if at each stage of divide-and-conquer either $\Gamma^+(u)$ or $\Gamma^-(u)$ is the empty set. Similarly, one can also have a divide-and-conquer algorithm to search for a king in T_n with a satisfactory average cost of $\Theta(n \log n)$ and the worst case cost of $n(n-1)/2$. The motivation of this paper is to provide alternative algorithms for the search of a king and a sorted sequence of kings, respectively, in the case that avoiding the above mentioned worst case is crucial. We achieve this goal with some sacrifice in the average performance.

Let $f(n)$ and $g(n)$ be the cost in the worst case for finding a king and a sorted sequence of kings, respectively, in T_n. Using an adversary argument proposed in this paper, we prove that

$$\frac{\sqrt{3}}{3}(n-1)^{3/2} - \frac{3}{2}n \le g(n) \le \frac{8\sqrt{2}}{3}n^{3/2} + 25n^{5/4}.$$

Therefore, the worst case asymptotic cost for finding a sorted sequence of kings in T_n is $\Theta(n^{3/2})$. We also prove that

$$\frac{3\sqrt[3]{2}}{4}(n-1)^{4/3} - \frac{3}{2}(n-1) \le f(n) \le \frac{4\sqrt{2}}{3}n^{3/2}.$$

That is, the worst case asymptotic cost for finding a king in T_n is $\Omega(n^{4/3})$ and $O(n^{3/2})$. Our proofs for both upper bounds provide algorithms for finding a king and a sorted sequence of kings, respectively, in T_n both with a complexity of $O(n^{3/2})$. To prove the lower bounds for $f(n)$ and $g(n)$, we design a pro-small-outdegree-strategy for an *adversary argument* [5]. We prove that if the adversary uses this strategy, no algorithms can succeed with cost smaller than the above mentioned lower bounds in the worst case.

The paper is organized as follows. Section 2 introduces the idea of improving the worst cast performance with the sacrifice of the average performance, presents a pro-small-outdegree-strategy for the use of an adversary argument, and uses both ideas to prove an upper bound and a lower bound, respectively, for the worst case cost for finding a king in T_n. Section 3 is devoted to the proof that the worst case asymptotic cost for finding a sorted sequence of kings in T_n is $\Theta(n^{3/2})$. Section 4 shows a connection between a sorted sequence of kings and a median order in a tournament. Section 5 concludes the paper and raises two open problems.

2. Worst case cost for finding a king. If one uses the divide-and-conquer algorithm introduced in the previous section to search for a king in T_n, the worst case happens when $\Gamma^+(u) = \emptyset$ at each stage of divide-and-conquer. So in order to avoid the worst case, one should carefully choose a pivot vertex u with $\Gamma^+(u)$ large enough at each stage of divide-and-conquer. If we merge this idea into the divide-and-conquer algorithm, we call it the *revised-divide-and-conquer algorithm*. For this purpose, we need the following lemma for the choice of a pivot vertex.

LEMMA 2 (Landau [6]). *Suppose $d_1^+ \le d_2^+ \le \cdots \le d_n^+$ is the outdegree sequence of T_n. Then, for each i,*

$$\frac{i-1}{2} \le d_i^+ \le \frac{n+i-2}{2}.$$

A simple idea to use a revised-divide-and-conquer algorithm to prove $f(n) = O(n^{3/2})$ is as follows. First, we can use n questions to obtain the orientation of all edges in a subtournament of roughly $\sqrt{2n}$ vertices. (Note that this part is the sacrifice of the average performance.) We choose a vertex with the maximum outdegree in this subtournament as the pivot vertex to apply Lemma 1. Then, at each stage of divide-and-conquer, at least $\sqrt{2n}/2$ vertices are eliminated with a total cost of $\binom{\sqrt{2n}}{2} + n - \sqrt{2n} < 2n$. Once the size of remaining vertices is sufficiently small, say $\sqrt{2n}/2$, use the direct method to find a king. Thus eliminating $n-1$ vertices with a king remaining costs at most $2n \cdot n/\left(\sqrt{2n}/2\right) = 2\sqrt{2}n^{3/2}$. We use a recursive relation to prove an improved coefficient for $n^{3/2}$ in the next theorem.

THEOREM 1. *One can find a king in T_n with cost at most $(4\sqrt{2})n^{3/2}/3$; that is,*

$$f(n) \le \frac{4\sqrt{2}}{3} n^{3/2}.$$

Proof. Let S be a subset of vertices in T_n with $|S| = \lceil \sqrt{2n} \rceil$. Let T_S be the subtournament of T_n induced by S. Then we can obtain the orientation of all edges in T_S with cost $\binom{\lceil \sqrt{2n} \rceil}{2}$. Let u be a vertex with the maximum outdegree within the subtournament T_S. By Lemma 2, the outdegree of u within T_S is

$$d^+(T_S, u) \ge \frac{|S| - 1}{2} = \frac{\lceil \sqrt{2n} \rceil - 1}{2}.$$

Next we can obtain the orientation of the edge between u and each $v \in V(T_n) \setminus S$ with cost $n - \lceil \sqrt{2n} \rceil$. Then

$$|\Gamma^-(u)| = n - 1 - |\Gamma^+(u)| \le n - 1 - d^+(T_S, u) \le n - \frac{\sqrt{2n}}{2}.$$

To find a king in T_n, by Lemma 1, it suffices to find a king in the subtournament of T_n induced by $\Gamma^-(u)$ with cost at most $f(|\Gamma^-(u)|)$. Thus

$$f(n) \le f(|\Gamma^-(u)|) + \binom{\lceil \sqrt{2n} \rceil}{2} + n - \lceil \sqrt{2n} \rceil \le f\left(n - \frac{\sqrt{2n}}{2}\right) + 2n - \frac{\sqrt{2n}}{2}.$$

Now we use induction to prove $f(n) \le 4\sqrt{2}n^{3/2}/3$ for all $n \ge 1$. It holds for $n = 1$ trivially. Suppose it holds for all cases less than n. Then

$$\begin{aligned} f(n) &\le f\left(n - \tfrac{\sqrt{2n}}{2}\right) + 2n - \tfrac{\sqrt{2n}}{2} \\ &\le \tfrac{4\sqrt{2}}{3}\left(n - \tfrac{\sqrt{2n}}{2}\right)^{3/2} + 2n - \tfrac{\sqrt{2n}}{2} \\ &\le \tfrac{4\sqrt{2}}{3}\left(n - \tfrac{\sqrt{2n}}{2}\right)\left(\sqrt{n} - \tfrac{\sqrt{2}}{4}\right) + 2n - \tfrac{\sqrt{2n}}{2} \\ &< \tfrac{4\sqrt{2}}{3} n^{3/2}. \qquad \square \end{aligned}$$

Remark 1. In the actual implementation of the revised-divide-and-conquer algorithm given in Theorem 1, an extra number of $\Theta(|S|)$ comparisons in determining the vertex with the maximum outdegree in T_s will be introduced at each recursive call in selecting a pivot vertex. However, since the total number of extra comparisons needed is $O(n)$, the algorithm will have a complexity of at most $4\sqrt{2}n^{3/2}/3 + O(n)$.

Remark 2. We note that if only $\sqrt{2n}/2$ vertices are eliminated in the first stage of divide-and-conquer, a complete subtournament of $\sqrt{2n}/2$ vertices remains. Then we can take advantage of having known all edges within this remaining subtournament to obtain a second complete subtournament of $\sqrt{2n}$ vertices with a cost of only $\binom{\sqrt{2n}}{2} - \binom{\sqrt{2n}/2}{2} \approx 3n/4$ for the second stage of divide-and-conquer. This shows that either more than $\sqrt{2n}/2$ vertices can be eliminated in the first stage of divide-and-conquer or the second stage of divide-and-conquer can be performed with cost less than $2n$. If this is taken into consideration recursively, a much longer and tedious estimate shows that $f(n) \le 2\sqrt{6}n^{3/2}/3 + o(n^{3/2})$.

In order to prove a lower bound for the worst case cost for finding a king in T_n, we use an adversary argument. Our idea is to design a strategy for the adversary to answer each question. The adversary chooses his/her answers to try to force the algorithm to work hard. Suppose $e_1, e_2, \ldots, e_l$, where $l = f(n)$ is a sequence of edges that we ask the adversary about regarding their orientation. Let $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_l$ be the answers of the adversary. We define a sequence of digraphs $G_0, G_1, \ldots, G_l$ as follows:

1. Let G_0 be the empty digraph with the same vertex set as T_n.
2. Let $G_i = G_{i-1} + \vec{e}_i$ for each $i = 1, 2, \ldots, l$; that is, G_i is obtained by adding the edge $\vec{e}_i$ to G_{i-1}.

Suppose v_i and w_i are the two vertices incident to e_i. We design the following strategy for the adversary to determine his/her answer to the question about the orientation of e_i:

1. Let $v_i \to w_i$ if $d^+(G_{i-1}, v_i) < d^+(G_{i-1}, w_i)$.
2. Let $w_i \to v_i$ if $d^+(G_{i-1}, v_i) > d^+(G_{i-1}, w_i)$.
3. Orientate e_i arbitrarily if $d^+(G_{i-1}, v_i) = d^+(G_{i-1}, w_i)$.

We call the above strategy the pro-small-outdegree-strategy and call G_l the digraph constructed by using the pro-small-outdegree-strategy. From now on we use $d(v)$ to denote $d^+(G_l, v)$ if there is no confusion from the context.

LEMMA 3. *Suppose v is a vertex in G_l constructed by using the pro-small-outdegree-strategy. Then, for any $S \subseteq \Gamma^+(v)$,*

$$\sum_{w \in S} d(w) \ge \binom{|S|}{2}.$$

Proof. We may order the vertices of $\Gamma^+(v) = \{w_i : 1 \le i \le d(v)\}$ according to the ordering of the questions whose answers are of the type "$v \to w$." Then $d(w_i) \ge i - 1$ by the pro-small-outdegree-strategy. Thus

$$\sum_{w \in S} d(w) \ge \sum_{1 \le i \le |S|} (i-1) = \binom{|S|}{2}. \qquad \square$$

LEMMA 4. *Suppose v is a vertex in G_l constructed by using the pro-small-outdegree-strategy. Then, for any $S \subseteq \Gamma^{++}(v)$,*

$$\sum_{w \in S} d(w) \ge d(v) \cdot \binom{|S|/d(v)}{2}.$$

Proof. Let $\Gamma^+(v) = \{w_i : 1 \le i \le d(v)\}$. Since $S \subseteq \Gamma^{++}(v)$, we can partition S into pairwise disjoint sets as follows:

1. $S = \cup_{i=1}^{d(v)} S_i$. (It is possible that some S_i may be empty.)
2. $S_i \subseteq \Gamma^+(w_i)$ for each i, $1 \le i \le d(v)$.

(Note that such a partition of S may not be unique since a vertex in S could be dominated by more than one vertex in $\Gamma^+(v)$.) By Lemma 3,

$$\sum_{w\in S} d(w) = \sum_{i=1}^{d(v)} \sum_{w\in S_i} d(w) \ge \sum_{i=1}^{d(v)} \binom{|S_i|}{2} \ge d(v) \cdot \binom{\sum |S_i|/d(v)}{2} = d(v) \cdot \binom{|S|/d(v)}{2},$$

where the last inequality holds since the function $\binom{x}{2}$ is concave upward. □

THEOREM 2. *No algorithm can find a king in T_n with cost less than $3\sqrt[3]{2}(n-1)^{4/3}/4 - 3(n-1)/2$ in the worst case.*

Proof. Suppose G_l ($l = f(n)$) is the digraph constructed by using the pro-small-outdegree-strategy. Let v be a king in G_l. Then $V(G_l) = \{v\} \cup \Gamma^+(v) \cup \Gamma^{++}(v)$, and so $|\Gamma^{++}(v)| = n - d(v) - 1$. By Lemmas 3 and 4,

$$\begin{aligned} f(n) &= l = |E(G_l)| = d(v) + \sum_{w\in\Gamma^+(v)} d(w) + \sum_{w\in\Gamma^{++}(v)} d(w) \\ &\ge d(v) + \binom{d(v)}{2} + d(v) \cdot \binom{(n-d(v)-1)/d(v)}{2} \\ &> \tfrac{1}{2}\left(\tfrac{(n-1)^2}{d(v)} + (d(v))^2 - 3n + 3\right) \\ &\ge \tfrac{3}{4}\sqrt[3]{2}(n-1)^{4/3} - \tfrac{3}{2}(n-1), \end{aligned}$$

where the last inequality follows from minimizing the function $(n-1)^2/x + x^2$. □

By combining Theorems 1 and 2, we conclude that the worst case asymptotic cost for finding a king in T_n is $\Omega(n^{4/3})$ and $O(n^{3/2})$.

3. Worst case cost for finding a sorted sequence of kings. If one uses the divide-and-conquer algorithm introduced in section 1 to search for a sorted sequence of kings in T_n, the worst case happens when either $\Gamma^+(u) = \emptyset$ or $\Gamma^-(u) = \emptyset$ at each stage of divide-and-conquer. Therefore, in order to avoid the worst case, one should carefully choose a pivot vertex u with both $\Gamma^+(u)$ and $\Gamma^-(u)$ large enough at each stage of divide-and-conquer. Similar to the proof of Theorem 1, we can first use n questions to obtain the orientation of all edges in a subtournament of roughly $\sqrt{2n}$ vertices. (Again note that this part is the sacrifice of the average performance.) We choose a vertex with the median outdegree in this subtournament as the pivot vertex u to apply Lemma 1. Then, by Lemma 2, each of $\Gamma^+(u)$ and $\Gamma^-(u)$ contains at least $\sqrt{2n}/4$ vertices.

THEOREM 3. *One can find a sorted sequence of kings in T_n with cost at most $(8\sqrt{2})n^{3/2}/3 + O(n^{5/4})$; that is,*

$$g(n) \le \frac{8\sqrt{2}}{3} n^{3/2} + cn^{5/4}$$

for some constant c; for example, one may choose $c = 25$.

Proof. Let S be a subset of vertices in T_n with $|S| = \lceil \sqrt{2n} \rceil + 2$. Let T_S be the subtournament of T_n induced by S. Then we can obtain the orientation of all edges in T_S with cost $\binom{\lceil\sqrt{2n}\rceil+2}{2}$. Let $d_1^+(T_S) \le d_2^+(T_S) \le \cdots \le d_{|S|}^+(T_S)$ be the outdegree sequence of vertices within T_S. Let u be a vertex in T_S such that $d^+(T_S, u) = d_t^+(T_S)$,

where $t = \left\lceil \left\lceil \sqrt{2n} \right\rceil /2 \right\rceil + 1$. (That is, u is a vertex with the median outdegree in T_S.) By Lemma 2, we have

$$d^+(T_S, u) \geq \frac{t-1}{2} \geq \frac{\sqrt{2n}}{4}$$

and

$$d^-(T_S, u) = |S| - 1 - d^+(T_S, u) \geq |S| - 1 - \frac{|S| + t - 2}{2} \geq \frac{\sqrt{2n}}{4}.$$

Next we can obtain the orientation of the edge between u and each $v \in V(T_n) \setminus S$ with cost $n - 2 - \left\lceil \sqrt{2n} \right\rceil$. Then

$$\frac{\sqrt{2n}}{4} \leq d^+(T_S, u) \leq |\Gamma^+(u)| \leq n - 1 - d^-(T_S, u) \leq n - \frac{\sqrt{2n}}{4}$$

and, similarly,

$$\frac{\sqrt{2n}}{4} \leq |\Gamma^-(u)| \leq n - \frac{\sqrt{2n}}{4}.$$

To find a sorted sequence of kings in T_n, by Lemma 1, it suffices to find sorted sequences of kings in both subtournaments of T_n induced by $\Gamma^+(u)$ and by $\Gamma^-(u)$, respectively, with total cost at most $g(|\Gamma^+(u)|) + g(|\Gamma^-(u)|)$. Thus

$$\begin{aligned} g(n) &\leq g(|\Gamma^+(u)|) + g(|\Gamma^-(u)|) + \binom{\left\lceil \sqrt{2n} \right\rceil + 2}{2} + n - 2 - \left\lceil \sqrt{2n} \right\rceil \\ &\leq g(|\Gamma^+(u)|) + g(|\Gamma^-(u)|) + 2n + \tfrac{3\sqrt{2n}}{2}. \end{aligned}$$

Now we use induction to prove $g(n) \leq h(n)$, where $h(n) = 8\sqrt{2}n^{3/2}/3 + 25n^{5/4}$, for all $n \geq 1$. It holds for $n = 1$ trivially. Suppose it holds for all cases less than n. Then

$$g(|\Gamma^+(u)|) + g(|\Gamma^-(u)|) \leq h(|\Gamma^+(u)|) + h(|\Gamma^-(u)|) \leq h\left(\frac{\sqrt{2n}}{4}\right) + h\left(n - \frac{\sqrt{2n}}{4}\right),$$

where the last inequality holds since the function $h(x) = 8\sqrt{2}x^{3/2}/3 + 25x^{5/4}$ is concave upward. Thus

$$\begin{aligned} g(n) &\leq h\left(\tfrac{\sqrt{2n}}{4}\right) + h\left(n - \tfrac{\sqrt{2n}}{4}\right) + 2n + \tfrac{3\sqrt{2n}}{2} \\ &\leq \tfrac{8\sqrt{2}}{3}\left(\tfrac{\sqrt{2n}}{4}\right)^{3/2} + 25\left(\tfrac{\sqrt{2n}}{4}\right)^{5/4} + \tfrac{8\sqrt{2}}{3}\left(n - \tfrac{\sqrt{2n}}{4}\right)\left(\sqrt{n} - \tfrac{\sqrt{2}}{8}\right) \\ &\quad + 25\left(n - \tfrac{\sqrt{2n}}{4}\right)\left(\sqrt[4]{n} - \tfrac{\sqrt{2}}{16\sqrt[4]{n}}\right) + 2n + \tfrac{3\sqrt{2n}}{2} \\ &< \tfrac{8\sqrt{2}}{3}n^{3/2} + 25n^{5/4}. \qquad \square \end{aligned}$$

Remark 3. In the actual implementation of the revised-divide-and-conquer algorithm given in Theorem 3, an extra number of $\Theta(|S|)$ comparisons will be introduced at each recursive call to select a median. The median can be determined using a linear selection algorithm [2]. It is still unknown exactly how many comparisons are needed to determine the median. Dor and Zwick [3] showed that the upper bound is

slightly less than $2.95|S|$ and the lower bound is slightly more than $2|S|$. Again, since the total number of extra comparisons needed is $O(n)$, the revised-divide-and-conquer algorithm will have a complexity of at most $8\sqrt{2}n^{3/2}/3 + 25n^{5/4} + O(n)$.

THEOREM 4. *No algorithm can find a sorted sequence of kings in T_n with cost less than $\sqrt{3}(n-1)^{3/2}/3 - 3n/2$ in the worst case.*

Proof. Suppose G_l ($l = g(n)$) is the digraph constructed by using the pro-small-outdegree-strategy. Suppose $u_1, u_2, \ldots, u_n$ form a sorted sequence of kings in G_l. Since $g(n) = l = |E(G_l)|$, it suffices to prove $|E(G_l)| \geq \sqrt{3}(n-1)^{3/2}/3 - 3n/2$. The proof is split into two cases.

Case 1. Suppose $d(u_i) \geq \sqrt{3(n-i)}/2$ for all i, $1 \leq i \leq n$. Then

$$|E(G_l)| = \sum_{i=1}^{n} d(u_i) \geq \frac{1}{2}\sum_{i=1}^{n}\sqrt{3(n-i)} \geq \frac{1}{2}\int_0^{n-1}\sqrt{3x}\,dx = \frac{\sqrt{3}}{3}(n-1)^{3/2}.$$

Case 2. Suppose $d(u_i) < \sqrt{3(n-i)}/2$ for some i, $1 \leq i \leq n$. Let t be the smallest i satisfying $d(u_i) < \sqrt{3(n-i)}/2$. Let $S_1 = \Gamma^+(u_t) \cap \{u_i : i \geq t+1\}$ and $S_2 = \Gamma^{++}(u_t) \cap \{u_i : i \geq t+1\}$. By the definition of a sorted sequence of kings, we know that S_1 and S_2 form a disjoint partition of $\{u_i : i \geq t+1\}$ and, hence, $|S_2| = n - t - |S_1|$. Since $|S_1| \leq d(u_t) < \sqrt{3(n-t)}/2$, by Lemma 4,

$$\begin{aligned}
\sum_{i=t+1}^{n} d(u_i) &\geq \sum_{u_i \in S_2} d(u_i) \\
&\geq d(u_t) \cdot \binom{(n-t-|S_1|)/d(u_t)}{2} \\
&\geq \tfrac{1}{2}\left(\tfrac{(n-t)^2}{d(u_t)} - \tfrac{2n|S_1|}{d(u_t)} - n\right) \\
&\geq \tfrac{1}{2}\left(\tfrac{2(n-t)^2}{\sqrt{3(n-t)}} - 3n\right) \\
&= \tfrac{\sqrt{3}}{3}(n-t)^{3/2} - \tfrac{3}{2}n.
\end{aligned}$$

Thus

$$\begin{aligned}
|E(G_l)| &\geq \sum_{i=1}^{t-1} d(u_i) + \sum_{i=t+1}^{n} d(u_i) \\
&\geq \tfrac{1}{2}\sum_{i=1}^{t-1}\sqrt{3(n-i)} + \tfrac{\sqrt{3}}{3}(n-t)^{3/2} - \tfrac{3}{2}n \\
&\geq \tfrac{\sqrt{3}}{2}\int_{n-t}^{n-1}\sqrt{x}\,dx + \tfrac{\sqrt{3}}{3}(n-t)^{3/2} - \tfrac{3}{2}n \\
&= \tfrac{\sqrt{3}}{3}(n-1)^{3/2} - \tfrac{3}{2}n. \qquad \square
\end{aligned}$$

By combining Theorems 3 and 4, we conclude that the worst case asymptotic cost for finding a sorted sequence of kings in T_n is $\Theta(n^{3/2})$.

4. Connection with median order. A sorted sequence of kings may also be viewed as a weak approximation for ranking players in a tournament. In general, the *tournament ranking problem* [8] is a difficult one without applausive solution. Suppose $u_1, u_2, \ldots, u_n$ is a ranking of players such that u_i is ranked in the ith place. For any

pair of players u_i, u_j with $i < j$, a *happiness* is an outcome that u_i beats u_j, while an *upset* is an outcome that u_j beats u_i. A ranking strategy introduced in [10] is to minimize the number of total upsets. A *median order* of a tournament is defined as a ranking of players with the minimum number of total upsets. Similarly, a median order of a digraph can be defined as an ordered list of vertices which induces an acyclic digraph with the maximum number of edges. It is known that determining a median order of a digraph is NP-complete and that the complexity for determining a median order for a tournament is still unknown [1]. Now suppose $u_1, u_2, \ldots, u_n$ form a median order for T_n. Havet and Thomassé [4] showed that u_1 is a king for T_n. By the definition of a median order, it is easy to see that $u_i, u_{i+1}, \ldots, u_n$ form a median order for the subtournament induced by $\{u_j : i \leq j \leq n\}$ for each $i \leq n$. These facts reveal the following connection between a median order and a sorted sequence of kings in a tournament.

THEOREM 5. *Any median order in a tournament is a sorted sequence of kings.*

Theorem 5 suggests that one does not have to check all $n!$ possible orderings of vertices in order to find a median order of T_n. Instead, one may narrow the search within all sorted sequences of kings.

5. Conclusion. In this paper, we have shown that the worst case asymptotic cost for finding a sorted sequence of kings in T_n is $\Theta(n^{3/2})$. We have also shown that the worst case asymptotic cost for finding a king in T_n is $\Omega(n^{4/3})$ and $O(n^{3/2})$. The lower bounds are derived by using an adversary argument called pro-small-outdegree-strategy proposed in this paper. In addition, we have provided a revised-divide-and-conquer algorithm that finds a sorted sequence of kings (including a king) with a cost of $\Theta(n^{3/2})$ in the worst case. It is still an open problem on exactly how many binary questions are needed in the worst case to determine a sorted sequence of kings in a tournament. Also it would be interesting to know the worst case asymptotic cost for finding a king in a tournament.

REFERENCES

[1] I. CHARON, A. GUÉNOCHE, O. HUDRY, AND F. WOIRGARD, *New results on the computation of median orders*, Discrete Math., 165/166 (1997), pp. 139–153.

[2] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction to Algorithms*, 2nd ed., MIT Press, Cambridge, MA, McGraw-Hill, Boston, 2001.

[3] D. DOR AND U. ZWICK, *Selecting the median*, SIAM J. Comput., 28 (1999), pp. 1722–1758.

[4] F. HAVET AND S. THOMASSÉ, *Median orders of tournaments: A tool for the second neighborhood problem and Sumner's conjecture*, J. Graph Theory, 35 (2000), pp. 244–256.

[5] D. E. KNUTH, *The Art of Computer Programming, Vol.* 3: *Sorting and Searching*, 2nd ed., Addison-Wesley, Reading, MA, 1998.

[6] H. G. LANDAU, *On dominance relations and the structure of animal societies.* III. *The condition for a score structure*, Bull. Math. Biophys., 15 (1953), pp. 143–148.

[7] W. LOU, J. WU, AND L. SHENG, *On the existence of a sorted sequence of kings in a tournament (abstract)*, in Proceedings of 31st Southeastern International Conference on Combinatorics, Graph Theory, and Computing, Boca Raton, FL, 2000.

[8] J. W. MOON, *Topics on Tournament*, Holt, Rinehart and Winston, New York, Montreal, London, 1968.

[9] K. B. REID AND L. W. BEINEKE, *Tournaments*, in Selected Topics in Graph Theory, L. W. Beineke and R. Wilson, eds., Academic Press, NY, 1979.

[10] P. SLATER, *Inconsistencies in a schedule of paired comparisons*, Biometrika, 48 (1961), pp. 303–312.

[11] D. B. WEST, *Introduction to Graph Theory*, 2nd ed., Prentice-Hall, Upper Saddle River, NJ, 2001.

[12] J. WU AND L. SHENG, *An efficient sorting algorithm for a sequence of kings in a tournament*, Inform. Process. Lett., 79 (2001), pp. 297–299.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1210–1235

AN OPTIMAL ALGORITHM FOR CHECKING REGULARITY*

Y. KOHAYAKAWA†, V. RÖDL‡, AND L. THOMA§

Abstract. We present a deterministic algorithm $\mathcal{A}$ that, in $O(m^2)$ time, verifies whether a given m by m bipartite graph G is *regular*, in the sense of Szemerédi [*Regular partitions of graphs*, in Problèmes Combinatoires et Théorie des Graphes (Orsay, 1976), Colloques Internationaux CNRS 260, CNRS, Paris, 1978, pp. 399–401]. In the case in which G is *not* regular enough, our algorithm outputs a *witness* to this irregularity. Algorithm $\mathcal{A}$ may be used as a subroutine in an algorithm that finds an ε-regular partition of a given n-vertex graph Γ in time $O(n^2)$. This time complexity is optimal, up to a constant factor, and improves upon the bound $O(M(n))$, proved by Alon et al. [*The algorithmic aspects of the regularity lemma*, J. Algorithms, 16 (1994), pp. 80–109], where $M(n) = O(n^{2.376})$ is the time required to square a 0–1 matrix over the integers.

Our approach is elementary, except that it makes use of linear-sized expanders to accomplish a suitable form of deterministic sampling.

Key words. Szemerédi's regularity lemma, quasi-randomness, deterministic sampling, expander graphs, regular pairs

AMS subject classifications. 05C35, 05C85, 05C99, 68R10

DOI. 10.1137/S0097539702408223

1. Introduction and the main result.

Szemerédi's regularity lemma [31] is a fundamental result in graph theory (see [25] for an excellent survey). Roughly speaking, this lemma states that any graph admits a partition of its vertex set so that most pairs induce "pseudorandom" or *regular* bipartite graphs. The original proof of the regularity lemma was nonconstructive, but Alon et al. [1, 2] succeeded in developing a fast deterministic algorithm for finding such a partition. Many of the existential results based on the regularity lemma could then be turned into algorithmic results. The algorithm in [1, 2] finds a regular partition of an n-vertex graph in $O(M(n))$ deterministic time, where $M(n) = O(n^{2.376})$ (see [11]) is the time required to square a 0–1 matrix over the integers. More recently, Frieze and Kannan [18] (see also [19]) showed that *sampling* can be used to develop a $O(n)$ time *randomized* algorithm that, given an n-vertex graph G, outputs a partition for G that is regular with high probability.

In both algorithms above (and in all algorithms for variants of the regularity lemma), the main algorithmic problem is to decide whether a given m by m bipartite graph G is regular; if G is *not* regular, we are required to find a "witness" for this irregularity. In this paper, we present a *deterministic* algorithm that solves this problem in $O(m^2)$ time. Given our algorithm, one can derive in a standard way

*Received by the editors May 23, 2002; accepted for publication (in revised form) April 24, 2003; published electronically August 6, 2003. This research was supported by a CNPq/NSF INT–0072064 cooperative grant. Some of the results in this paper appeared in an extended abstract in *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, CA, 2002.
http://www.siam.org/journals/sicomp/32-5/40822.html

†Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, 05508–090 São Paulo, SP, Brazil (yoshi@ime.usp.br). This author was partially supported by MCT/CNPq through ProNEx Programme (Proc. CNPq 664107/1997–4), by CNPq (Proc. 300334/93–1 and Proc. 468516/2000–0), and by FAPESP (Proc. 96/04505–2).

‡Department of Mathematics and Computer Science, Emory University, Atlanta, GA 30322 (rodl@mathcs.emory.edu). This author was partially supported by NSF grant 0071261.

§Department of Mathematics, University of Rhode Island, Kingston, RI 02881 (thoma@math.uri.edu). This author was partially supported by NSF grants DMS–0301228 and DMS–9970622.

an algorithm for Szemerédi's regularity lemma that finds a regular partition of an n-vertex graph in time $O(n^2)$. A key feature of our approach lies in the use of linear-sized expanders for carrying out a certain procedure that may be thought of as *deterministic sampling*.

1.1. The main result. Let $G = (A, B; E)$ be a bipartite graph. If $\emptyset \neq U \subset A$, $\emptyset \neq V \subset B$, the *density* of (U, V) in G is $d(U, V) = e(U, V)/|U||V|$, where we write $e(U, V) = e_G(U, V)$ for the number of edges with one endpoint in U and the other endpoint in V. For $\varepsilon > 0$, we say that G is *ε-regular* if, for all $U \subset A, |U| \geq \varepsilon|A|$, and $V \subset B, |V| \geq \varepsilon|B|$, we have

$$|d(U, V) - d(A, B)| \leq \varepsilon \,. \tag{1.1}$$

In case G is not ε-regular and a certain pair (U, V) certifies this fact, then we say that (U, V) is a *witness* to the *ε-irregularity* of G.

Let $\Gamma = (V, E)$ be a graph. A partition $(V_i)_{i=0}^k$ of the vertex set V, $V = \bigcup_{i=0}^k V_i$, is said to be an *equitable partition* (with the exceptional class V_0) if $|V_1| = \cdots = |V_k|$. If $V = \bigcup_{i=0}^k V_i$ is an equitable partition of V such that the size of the exceptional class $|V_0| \leq \varepsilon n$ and at least $(1 - \varepsilon)\binom{k}{2}$ pairs (V_i, V_j), where $1 \leq i < j \leq k$, are ε-regular, we say that the partition $(V_i)_{i=0}^k$ is an *ε-regular partition*. We say that a pair (U, W) is *ε-regular* if the bipartite graph induced by (U, W) is ε-regular.

Szemerédi's remarkable result may be stated as follows.

THEOREM 1.1. *For any $\varepsilon > 0$ and any $k_0 \geq 1$, there is $K_0(\varepsilon, k_0)$ such that any graph Γ admits an ε-regular partition into k parts for some k satisfying $k_0 \leq k \leq K_0(\varepsilon, k_0)$.*

Alon et al. [1, 2] proved the following algorithmic version of Theorem 1.1.

THEOREM 1.2. *There is a deterministic algorithm $\mathcal{A}_0$ that, given $\varepsilon > 0$, $k_0 \geq 1$, and Γ, produces an ε-regular partition for Γ into k parts for some k satisfying $k_0 \leq k \leq K_0'$, where $K_0' = K_0'(\varepsilon, k_0)$ depends only on ε and k_0. Moreover, algorithm $\mathcal{A}_0$ runs in time $O(M(n)) = O(n^{2.376})$ if Γ has n vertices.*

Consider now the following closely related decision problem.

PROBLEM 1.3. Given a graph G, a pair (U, W) of nonempty, pairwise disjoint sets of vertices of G, and a positive ε, decide whether (U, W) is ε-regular with respect to G.

As it turns out, *the problem above is coNP-complete* [1, 2]. However, as observed already in [1, 2], to prove Theorem 1.2 it suffices to solve an *approximate* version of the decision problem above. For instance, the following result [15] suffices.

THEOREM 1.4. *There exists an algorithm $\mathcal{A}_1$ for which the following holds. When $\mathcal{A}_1$ receives as input an $\varepsilon > 0$ and a bipartite graph $G = (A, B; E)$ with $|A| = |B| = m \geq (2/\varepsilon)^5$, it either correctly asserts that G is ε-regular or else it returns a witness for the ε'-irregularity of G, where $\varepsilon' = \varepsilon'_{\mathcal{A}_1}(\varepsilon) = \varepsilon^5/16$. The running time of $\mathcal{A}_1$ is $O(M(m)) = O(m^{2.376})$.*

(See Frieze and Kannan [20] for a somewhat different approach to verifying regularity, based on singular values of matrices.) Our main result is an improvement of Theorem 1.4 above and may be stated as follows.

THEOREM 1.5 (the main result). *There exists an algorithm $\mathcal{A}$ for which the following holds. When $\mathcal{A}$ receives as input an $\varepsilon > 0$ and a bipartite graph $G = (A, B; E)$ with $|A| = |B| = m \geq m_0(\varepsilon)$, it either correctly asserts that G is ε-regular or else it returns a witness for the ε'-irregularity of G, where $\varepsilon' = \varepsilon'_{\mathcal{A}}(\varepsilon) = \varepsilon^{20}/10^{24}$. The running time of $\mathcal{A}$ is $O(m^2)$.*

We describe our algorithm $\mathcal{A}$ in section 3.1. Deriving an algorithm for the regularity lemma from Theorem 1.5 is standard (cf. section 3.2).

COROLLARY 1.6. *There is a deterministic algorithm $\mathcal{A}_0'$ that, given $\varepsilon > 0$, $k_0 \geq 1$, and a graph Γ, produces an ε-regular partition for Γ into k parts for some k satisfying $k_0 \leq k \leq K_0''$, where $K_0'' = K_0''(\varepsilon, k_0)$ depends only on ε and k_0. Moreover, algorithm $\mathcal{A}_0'$ runs in time $O(n^2)$ if Γ has n vertices.*

Clearly, algorithm $\mathcal{A}_0'$ above has optimal time complexity, up to the constant implicit in the big-O notation. In [1, 2], several algorithmic consequences are derived from Theorem 1.2. In the examples presented there, the time complexity of the algorithms is $O(M(n))$. *Using $\mathcal{A}_0'$ from Corollary 1.6, one obtains algorithms with optimal time complexity $O(n^2)$.* We also observe that a similar improvement may be obtained from Theorem 1.5 for the subgraph counting algorithm given by Duke, Lefmann, and Rödl [15].

Let us also mention that an important variant of the regularity lemma, suitable for finding *induced subgraphs*, was recently discovered by Alon et al. [3, 4] in the context of *property testing* (see, e.g., [21] and [22, 23]). In the applications of their regularity lemma in [3, 4] the authors do not need algorithms for finding their regular partitions; however, they observe that an algorithmic version of their lemma readily follows from results such as Theorem 1.4. Again, an $O(n^2)$ time algorithm follows immediately from Theorem 1.5.

Finally, we mention that one may prove a "nonbipartite version" of Theorem 1.5. This variant of our result implies that one may check in time $O(n^2)$ whether a given n-vertex graph Γ is *quasi-random* in the sense of Chung, Graham, and Wilson [10]. Moreover, if Γ is not quasi-random, then our algorithm will produce a suitable witness proving this, i.e., an induced subgraph with $\Omega(n)$ vertices whose density deviates substantially from the density of Γ (see section 1.3.5 for more details).

1.2. Local conditions for regularity. One may prove Theorem 1.4 by considering a certain "local condition" on $G = (A, B; E)$ that is essentially equivalent to the regularity of G. For simplicity, let us suppose that G is degree-regular. The condition is simply that the following inequality should hold:

$$\sum_{x,\, y \in A} \left| d_G(x,y) - p(G)^2 m \right| \leq \delta p(G)^2 m^3, \tag{1.2}$$

where $d_G(x,y) = |N_G(x) \cap N_G(y)|$ is the so-called *codegree* of x and y, and $p(G) = |E|/|A||B| = |E|/m^2$, with $m = |A| = |B|$, is the density of G. Clearly, inequality (1.2) may be checked in $O(m^3)$ time, and, in fact, using fast matrix multiplication, one may verify (1.2) in $O(M(m)) = O(m^{2.376})$ time. The precise meaning of the equivalence of the ε-regularity of G and the validity of (1.2) is as follows: for all $\varepsilon > 0$ there is $\delta > 0$ such that if (1.2) holds, then G is ε-regular. Moreover, for all $\delta > 0$, there is $\varepsilon' > 0$ such that if (1.2) fails, then G is not ε'-regular, and, in fact, a witness to this ε'-irregularity may be constructed explicitly in the same deterministic time. Some of the ideas described in this paragraph have appeared in the literature under many guises. (For a detailed discussion on the combinatorial aspects, see [24]; for applications of these ideas in theoretical computer science, see [28] and the references therein.) Basically, we are obtaining a somewhat surprising amount of information from "pairwise independence." We do not go into the details here.

The key idea in the proof of Theorem 1.5 is that we may restrict the sum in (1.2) to a *small*, *randomly selected* collection of pairs $\{x, y\}$ (and, naturally, scale down the right-hand side). This would not be so satisfactory, as we would have a randomized

procedure: *we in fact show that we may achieve the same effect by "deterministic sampling," using the edge set of a linear-sized expander J* (see the definition of property $\mathcal{P}(J,\delta)$ in section 3.1.2).

1.3. Algorithmic applications. As mentioned above, algorithms $\mathcal{A}$ and $\mathcal{A}'_0$ immediately imply improvements on deterministic algorithms that are based on Szemerédi's regularity lemma. Here we present a few typical examples of such algorithms. For more algorithmic applications of the regularity lemma see [2] and [15].

1.3.1. MAXCUT in dense graphs. There has been considerable interest in the following computational problem recently.

PROBLEM 1.7 (MAXCUT). Given a graph G, find a partition (U,W) of the vertex set of G so that the number of edges $e(U,W)$ between U and W is maximum.

It follows from the algorithmic version of the regularity lemma that one may design a polynomial time approximation scheme (PTAS) for MAXCUT if the input graphs G are restricted to dense graphs. Let us be more precise.

Let α be a fixed positive real. In this section, we consider only graphs G with edge density $e(G)\binom{|V(G)|}{2}^{-1} \geq \alpha$. Theorem 1.2 implies the following result: for any ε and $\alpha > 0$, there exist a constant $C(\varepsilon,\alpha)$ and a *deterministic* algorithm $\mathcal{A}_{\rm MC}$ so that, given an n-vertex graph G with edge density $\geq \alpha$, algorithm $\mathcal{A}_{\rm MC}$ returns a solution (U',W') for MAXCUT such that

$$e(U',W') \geq (1-\varepsilon)e(U^*,W^*), \tag{1.3}$$

where (U^*,W^*) is an optimal solution for G. Furthermore, the running time of $\mathcal{A}_{\rm MC}$ is $\leq C(\varepsilon,\alpha)M(n)$.

Algorithm $\mathcal{A}_{\rm MC}$ uses algorithm $\mathcal{A}_0$ in Theorem 1.2 as a subroutine; we may use, instead, algorithm $\mathcal{A}'_0$ in Corollary 1.6: let $\mathcal{A}'_{\rm MC}$ be the corresponding algorithm.

THEOREM 1.8. *On input G as above, the deterministic algorithm $\mathcal{A}'_{\rm MC}$ produces a partition (U,W) satisfying* (1.3) *in time $\leq C'(\varepsilon,\alpha)n^2$, where $C'(\varepsilon,\alpha)$ is a constant that depends only on ε and α.*

We remark that a *randomized* algorithm with time complexity $O(n^2)$ was already given by de la Vega [14]. For related results concerning randomized algorithms, the reader is referred to Frieze and Kannan [18, 19].

1.3.2. The quasi-Ramsey number and maximum acyclic subgraphs. Let $f : E(K_n) \to \{-1,1\}$ be a function and set $f(S) = \sum_{e\in\binom{S}{2}} f(e)$, where $S \subseteq [n]$. Here, as usual, K_n stands for the complete graph on n vertices and $\binom{S}{2}$ denotes the set of all pairs on the set S.

The *quasi-Ramsey number* $g(n)$ is defined as

$$g(n) = \min_f \max_{S\subseteq[n]} |f(S)| .$$

Erdős and Spencer [17] showed that

$$c_1 n^{3/2} \leq g(n) \leq c_2 n^{3/2}$$

for some absolute constants c_1 and $c_2 > 0$.

Let T_n be a tournament and P_n a transitive tournament both on n vertices. Set $|T_n \cap P_n|$ to be the number of common oriented arcs of T_n and P_n. The tournament ranking function $h(n)$ is defined by

$$h(n) = \min_{T_n} \max_{P_n} |T_n \cap P_n|;$$

i.e., $h(n)$ is the maximum number of edges one can choose in any tournament of order n without creating an oriented cycle. Spencer [29, 30] showed that

$$c_1 n^{3/2} \le h(n) - \frac{1}{2}\binom{n}{2} \le c_2 n^{3/2}$$

for some absolute constants c_1 and $c_2 > 0$.

A polynomial time approximation scheme (PTAS) for a maximization problem is a family of algorithms $\{\mathcal{S}_\varrho \colon 0 < \varrho < 1\}$ as follows. For any given $0 < \varrho < 1$, algorithm $\mathcal{S}_\varrho$ runs in polynomial time and finds a solution whose value is at least $(1-\varrho)\,\mathrm{OPT}$, where OPT in the optimal value. Using a constructive version of the regularity lemma, Czygrinow, Poljak, and Rödl [13, Theorem 3] designed a PTAS for the "dense" quasi-Ramsey problem and for tournament ranking.

For $f : E(K_n) \to \{-1, 1\}$ set $\mathrm{OPT}(f) = \max_{S \subseteq [n]} |f(S)|$. Our algorithm for the regularity lemma implies an improvement on the time complexity of the PTAS designed in [13].

THEOREM 1.9. *Let $c > 0$ be fixed. For every $0 < \varrho < 1$, there is a $O(n^2)$ time algorithm that constructs a set S such that*

$$|f(S)| \ge (1-\varrho)\,\mathrm{OPT}(f)$$

for any instance $f \colon E(K_n) \to \{-1, 1\}$ with $\mathrm{OPT}(f) \ge cn^2$.

Now, let $\mathrm{OPT}(T_n) = \max_{P_n} |T_n \cap P_n|$, where T_n is a tournament. Algorithm $\mathcal{A}_0'$ in Corollary 1.6 improves the time complexity of the PTAS designed in [13] to $O(n^2)$.

THEOREM 1.10. *Let $0 < \varrho < 1$. Then there is a $O(n^2)$ time algorithm that, given a tournament T_n, constructs an ordering σ of the vertices of T_n so that at least $(1-\varrho)\,\mathrm{OPT}(T_n)$ arcs agree with σ.*

1.3.3. Robustly high-chromatic graphs. Goldreich, Goldwasser, and Ron [22, 23] have recently initiated a systematic study of *property testing* for combinatorial structures. Roughly speaking, in property testing one has a property P of interest, and one is given an object X and a real number $\varepsilon > 0$. The task is then to decide whether X has P or if it is ε-far from any object Y having P (we suppose our objects are in some metric space). Furthermore, we wish to perform this test extremely quickly; typically, the tests examine a small random portion of X and distinguish between the two cases above with high probability of success. Thus, in an appropriate computational model, the tests have sublinear complexity (see [22, 23] for details).

A graph property P that has been proved to be testable [22, 23] is the property of having chromatic number at least k for any fixed k. This result was in fact implicit in [16], where the regularity lemma is used to prove that "robustly high-chromatic graphs" admit witnesses of bounded size. Indeed, the existential result in Theorem 1.11 below was proved in [16]. The algorithmic result in Theorem 1.11, but with time complexity $O(M(n))$, was proved in [1, 2].

THEOREM 1.11. *Let $k \ge 3$ be an integer, and let $\varepsilon > 0$ be a real constant. Then there exist integers $n_0 = n_0(k, \varepsilon)$ and $f = f(k, \varepsilon)$ and a constant $\nu = \nu(k, \varepsilon) > 0$ such that if $G = (V, E)$ is a graph with $n \ge n_0$ vertices, then either*

(i) *there exists a graph H on $h \le f$ vertices with chromatic number $\chi(H) \ge k$ that occurs in G at least νn^h times as a subgraph or else*

(ii) *there exists a set $E' \subseteq E$ with $|E'| \le \varepsilon n^2$ such that the subgraph $G' = (V, E \setminus E')$ satisfies $\chi(G') < k$.*

Furthermore, there is a deterministic algorithm that receives as input a graph $G = (V, E)$ *as above and, in time* $O(n^2)$*, outputs either a graph* H *as in* (i) *or else it outputs a set of edges* E' *as in* (ii)*, together with a proper coloring* $\Delta\colon V \to \{1, \ldots, k-1\}$ *of the subgraph* G'*.*

The approach in [22, 23] does not use the regularity lemma and implies the existential part of Theorem 1.11. Moreover, that approach also gives a randomized, polynomial time algorithm for the constructive part of Theorem 1.11.

Finally, we mention that Czumaj and Sohler [12] have recently proved that the property of having chromatic number at least k is also testable for hypergraphs.

1.3.4. Counting subgraphs. In this section, we describe an algorithm for approximately counting small subgraphs in large graphs. This algorithm will also be an application of algorithm $\mathcal{A}_0'$ from Corollary 1.6.

We need to introduce some notation. We shall follow [15]. Let $G = (V, E)$ be a graph on n vertices whose vertex set $V = \{v_1, \ldots, v_n\}$ is ordered by $v_1 < \cdots < v_n$. Let the set $W = \{w_1, \ldots, w_k\}$ be ordered by $w_1 < \cdots < w_k$. We say that a graph H with vertex set W is *order isomorphic* to an induced subgraph H' of G if there exists an isomorphism $\phi : H \to H'$ with the property that for each i and j, if $w_i < w_j$, then $\phi(w_i) < \phi(w_j)$. Let $H_1, \ldots, H_t$, where $t = 2^{\binom{k}{2}}$, be the list of all graphs on the set W, and let $\sigma_k(G) = (h_1, \ldots, h_t)$ be the t-dimensional vector in which each h_i is the number of induced subgraphs of G to which H_i is order isomorphic.

The following proposition asserts the existence of a certain type of approximation algorithm for the vector $\sigma_k(G)$. For more details, see [15].

THEOREM 1.12. *Let* $k \geq 3$ *be a fixed integer, and suppose* $\delta > 0$ *is a fixed real. There is an algorithm that, on input* G*, a labeled, ordered graph on* n *vertices, produces an approximation* $\overline{\sigma}_k(G) = (\overline{h}_1, \ldots, \overline{h}_t)$ *to the vector* $\sigma_k(G) = (h_1, \ldots, h_t)$ *with the property that*

$$|h_i - \overline{h}_i| \leq \delta \binom{n}{k}$$

for all $1 \leq i \leq t$*. This algorithm runs in time* $O(n^2)$*.*

In [15], the authors consider the problem of approximating $\sigma_k(G)$ for $k = k(n)$ slowly increasing functions of n. Our results may be used to improve on the time complexity of the algorithms given in [15] for such $k = k(n)$, but we shall not go into the details.

1.3.5. Checking quasi-randomness. Thomason [32] and Chung, Graham, and Wilson [10] initiated a systematic study of *quasi-random* properties of graphs: these are properties that are shared by almost all graphs and are in fact *deterministically* asymptotically equivalent; i.e., if a large graph has one of these properties, then it in fact has all of them.

The investigation of quasi-randomness in combinatorics turned out to be a very rich line of research, as shown in the series of papers by Chung and Graham on the subject (for recent developments, see [9] and the references therein). Besides graphs, other combinatorial structures such as tournaments, set-systems, and subsets of $\mathbb{Z}/n\mathbb{Z}$ have been studied from this perspective (see [6, 7, 8]). Finally, we mention that applications of some of the underlying ideas in this area have occurred in the literature in different contexts; the interested reader is referred to [5, Chapter 9] and [24].

In this section, we shall consider the computational problem of determining whether or not a given graph is quasi-random. We are also interested in an additional requirement: in the case in which the input graph is *not* quasi-random, a "witness" to certify this fact should be efficiently produced.

We shall use the following definition.

DEFINITION 1.13. *Let reals $0 < \varepsilon \le 1$ and $0 < \delta \le 1$ be given. We shall say that a graph G is* $(1/2, \varepsilon, \delta)$-quasi-random *if, for all U, $W \subset V(G)$ with $U \cap W = \emptyset$ and $|U|$, $|W| \ge \delta n$, we have*

$$\left| e_G(U,W) - \frac{1}{2}|U||W| \right| \le \frac{1}{2}\varepsilon |U||W|.$$

In [24] the authors consider a new quasi-random property to develop an algorithm for testing quasi-randomness. Let G be a graph on n vertices, and let J be a (ϱ, L)-uniform graph on the same vertex set (for the definition, see section 2). To state the results we need to introduce some notation. For a vertex i of G we set $N(i)$ to be its neighborhood. Further, we write $N(i) \triangle N(j)$ for the symmetric difference of the sets $N(i)$ and $N(j)$.

To decide about the quasi-randomness of G we introduce the following couple of properties. Let $0 < \varepsilon, \delta \le 1$ be real numbers. We say that G satisfies property $\mathcal{T}_{\triangle}(J, \varepsilon)$ if we have

$$\sum_{\{i,j\} \in E(J)} \left| |N(i) \triangle N(j)| - \frac{1}{2}n \right| \le \frac{1}{2}\varepsilon n e(J) \, .$$

Note that this property is closely related to our property $\mathcal{P}$ introduced below. Similarly, we say that G satisfies property $\mathcal{T}'_{\triangle}(J, \gamma, \varepsilon)$ if the inequality

$$\left| |N(i) \triangle N(j)| - \frac{1}{2}n \right| \le \frac{1}{2}\varepsilon n$$

fails for at most $\gamma e(J)$ edges $\{i,j\} \in E(J)$.

The following two characterization theorems are proved in [24, Theorems 56 and 57].

THEOREM 1.14. *For any $0 < \varepsilon, \delta \le 1$ and any L, there exist $\varepsilon_0 = \varepsilon_0(\varepsilon, \delta, L) > 0$ and $r_0 = r_0(\varepsilon, \delta, L) \ge 1$ for which the following holds. Let G and J be two graphs on the same vertex set of n vertices. Assume further that J is a (ϱ, L)-uniform graph with the average degree $r = \varrho n \ge r_0$. Then, if G satisfies the property $\mathcal{T}_{\triangle}(J, \varepsilon')$ for some $0 < \varepsilon' \le \varepsilon_0$, then G is $(1/2, \varepsilon, \delta)$-quasi-random.*

THEOREM 1.15. *For any $0 < \gamma, \varepsilon \le 1$ and any L, there exist $\varepsilon_1 = \varepsilon_1(\gamma, \varepsilon, L) > 0$, $\delta_1 = \delta_1(\gamma, \varepsilon, L) > 0$, $r_1 = r_1(\gamma, \varepsilon, L) \ge 1$, and $N_1 = N_1(\gamma, \varepsilon, L) \ge 1$ for which the following holds. Let G and J be two graphs on the same vertex set of $n \ge N_1$ vertices. Assume further that J is a (ϱ, L)-uniform graph with the average degree $r = \varrho n \ge r_1$. Then, if G is $(1/2, \varepsilon', \delta')$-quasi-random for some $0 < \varepsilon' \le \varepsilon_1$ and $0 < \delta' \le \delta_1$, then property $\mathcal{T}'_{\triangle}(J, \gamma, \varepsilon)$ holds for G.*

It is straightforward to see that the properties $\mathcal{T}$ and $\mathcal{T}'$ can be checked in $O(n^2)$ deterministic time. Moreover, if a graph $\mathcal{G}$ does not satisfy property $\mathcal{T}'_{\triangle}(J, \gamma, \varepsilon)$, then one can, using the ideas from our present paper, construct a witness for the non–quasi-randomness of G in $O(n^2)$ time.

2. Preliminaries. In this section, we discuss some basic properties and the algorithmic construction of certain very well known random looking graphs.

2.1. (ϱ, L)-uniformity. Let $0 < \varrho \le 1$ and $L > 0$ be fixed. We say that a graph J on m vertices is (ϱ, L)-*uniform* if, for any U, $W \subset V(J)$ with $U \cap W = \emptyset$, we have

$$\big|e_J(U,W) - \varrho|U||W|\big| \le L\sqrt{r|U||W|}, \tag{2.1}$$

where $r = \varrho m$. The following lemma is immediate.

LEMMA 2.1. *Let $R = (V, E)$ be a (ϱ, L)-uniform, m-vertex graph and let $\emptyset \ne A \subset V$ be given. Put $J = R[A]$. Then J is an (ϱ, L')-uniform graph with $L' = L\sqrt{m/|A|}$.*

Notation 2.2. We use the following nonstandard notation: we write $O_1(x)$ for any term y such that $|y| \le x$.

We shall need estimates on the number of edges induced on subsets of (ϱ, L)-uniform graphs. Below, if Γ is a graph, we write $e(\Gamma)$ for the number of edges in Γ.

LEMMA 2.3. *Let $J = (V, E)$ be a (ϱ, L)-uniform graph, and let $S \subseteq V$ be a nonempty subset of vertices of J. Then*

$$\begin{aligned} e(J[S]) &= \varrho\binom{|S|}{2} + O_1\big(Lr^{1/2}(|S|+1)\big) \\ &= \varrho\frac{|S|^2}{2} + O_1\big(2Lr^{1/2}|S|\big), \end{aligned} \tag{2.2}$$

where $r = \varrho|V|$.

Proof. Put $s = |S|$. Note that, for any $1 \le t < s$, we have $2e(S)\binom{s-2}{t-1} = \sum_T e(T, S \setminus T)$, where the sum is extended over all $T \subset S$ with $|T| = t$. Thus

$$e(S) = \frac{1}{2}\binom{s}{t}\binom{s-2}{t-1}^{-1}\Big\{\varrho|T||S \setminus T| + O_1\big(L\{rt(s-t)\}^{1/2}\big)\Big\}$$

for any $1 \le t < s$. We use this relation with $t = \lfloor s/2 \rfloor$. Note that

$$\binom{s}{\lfloor s/2 \rfloor}\binom{s-2}{\lfloor s/2 \rfloor - 1}^{-1} = \frac{s(s-1)}{\lfloor s/2 \rfloor \lceil s/2 \rceil} \le 4,$$

and so

$$e(S) = \varrho\binom{s}{2} + O_1\big(2L\{r\lfloor s/2 \rfloor \lceil s/2 \rceil\}^{1/2}\big) = \varrho\binom{s}{2} + O_1\big(Lr^{1/2}(s+1)\big),$$

and the result follows. □

In what follows, the following simple consequences of Lemma 2.3 will be useful.

LEMMA 2.4. *Let $\eta > 0$ and $L > 0$ be given. Then there is an $\overline{r} = \overline{r}(\eta, L)$ such that any m-vertex (ϱ, L)-uniform graph J with $\varrho m \ge \overline{r}$ has the two properties below.*

(a) *If $S \subset V(J)$ is such that $|S| = \nu m \ge \eta m$, then*

$$e(J[S]) = (1 + O_1(\eta))\nu^2 e(J)\,. \tag{2.3}$$

(b) *If $S \subset V(J)$ is such that $|S| < \eta m$, then*

$$e(J[S]) < 2\eta^2 e(J)\,. \tag{2.4}$$

2.2. Auxiliary results on expander graphs. The celebrated Ramanujan graphs of Lubotzky, Phillips, and Sarnak [26, 27] are explicitly constructible examples of linear-sized $(\varrho, 2)$-uniform graphs. We shall make crucial use of their construction.

The Ramanujan graphs $X^{p,q}$ constructed in [26, 27] depend on certain primes p and q, which have to satisfy certain simple arithmetical conditions. The graphs $X^{p,q}$ that we shall be interested in are $(p+1)$-regular and have $q(q^2-1)/2$ vertices. However, we shall need to construct linear-sized $(\varrho, O(1))$-uniform graphs with m vertices and average degree around r, where m and r are arbitrary integers (which we may assume to be large). The main result of this section, Lemma 2.5, asserts that this can be done efficiently. As the reader will see, we shall simply check that, given m and r, we may find suitable primes p and q so that an induced subgraph of $X^{p,q}$ will do.

LEMMA 2.5. *There exists an algorithm $\mathcal{E}$ satisfying the following properties. There is an absolute constant r_1 such that for all $r_0 \geq r_1$ there are constants $m_0 = m_0(r_0)$ and $C_0 = C_0(r_0)$ for which the following holds. Algorithm $\mathcal{E}$ receives as input integers $r_0 \geq r_1$ and $m \geq m_0 = m_0(r_0)$, and returns an adjacency list representation of a particular $(\varrho, 3)$-uniform graph J on m vertices with $r = \varrho m$ satisfying $r_0 \leq r \leq 2r_0$. Furthermore, algorithm $\mathcal{E}$ runs in time $\leq C_0 m (\log m)^2$.*

In the remainder of this section, we prove Lemma 2.5 for completeness.

2.2.1. Ramanujan graphs. Before we start with the proof of Lemma 2.5, we recall the construction of Lubotzky, Phillips, and Sarnak [26, 27].

As usual, in what follows, if a is an integer and p is a prime with a not divisible by p, the Legendre symbol $(\frac{a}{p})$ is defined as 1 if a is a quadratic residue modulo p and as -1 if a is a quadratic nonresidue modulo p. To describe the construction in [26, 27], let p and q be two unequal primes satisfying

$$p,\, q \equiv 1 \pmod 4 \tag{2.5}$$

and

$$\left(\frac{p}{q}\right) = 1. \tag{2.6}$$

We now let S and T be the following sets. Below, i is an arbitrary fixed integer such that $i^2 \equiv -1 \pmod q$. We let

$$\begin{aligned} S = \big\{(\alpha_0, \alpha_1, \alpha_2, \alpha_3) \in \mathbb{Z}^4 \colon\ & \alpha_0^2 + \alpha_1^2 + \alpha_2^2 + \alpha_3^2 = p \\ & \text{with } \alpha_0 > 0,\ \text{odd, and } \alpha_1, \alpha_2, \alpha_3 \text{ even}\big\}, \end{aligned}$$

$$T = \left\{ \begin{pmatrix} \alpha_0 + i\alpha_1 & \alpha_2 + i\alpha_3 \\ -\alpha_2 + i\alpha_3 & \alpha_0 - i\alpha_1 \end{pmatrix} : (\alpha_0, \alpha_1, \alpha_2, \alpha_3) \in S \right\}. \tag{2.7}$$

We now consider $\mathrm{PSL}(2, \mathbb{Z}/q\mathbb{Z})$ (the projective special linear group), which consists of the 2×2 matrices over $\mathbb{Z}/q\mathbb{Z}$ whose determinants are nonzero quadratic residues mod p, quotiented out by the equivalence relation that makes two such matrices equivalent if one is a nonzero scalar multiple of the other.

It will be convenient to observe that each element of $\mathrm{PSL}(2, \mathbb{Z}/q\mathbb{Z})$ (i.e., each equivalence class) may be represented by a matrix whose second row is either $(0, 1)$ or $(1, x)$, where x is some arbitrary element of $\mathbb{Z}/q\mathbb{Z}$. The existence of this simple "canonical representation" for the elements of $\mathrm{PSL}(2, \mathbb{Z}/q\mathbb{Z})$ will be helpful below.

Observe that if we consider the entries of the matrices in T modulo q, we get 2×2 matrices over $\mathbb{Z}/q\mathbb{Z}$, with determinant $p \pmod q$, which is a nonzero quadratic residue

modulo q (cf. (2.6)). By a well-known result of Jacobi and some simple arguments, one may check that there are $p+1$ elements in T and that they are all distinct in $\mathrm{PSL}(2,\mathbb{Z}/q\mathbb{Z})$.

The graph $X^{p,q}$ constructed in [26, 27] is the Cayley graph of $\mathrm{PSL}(2,\mathbb{Z}/q\mathbb{Z})$ relative to the set T. The vertices of $X^{p,q}$ are the elements of $\mathrm{PSL}(2,\mathbb{Z}/q\mathbb{Z})$, and the edge set of $X^{p,q}$ is so that $\{x,y\}$ is an edge of $X^{p,q}$ if and only if there is a $t\in T$ such that $x=ty$ (one may check that this is a symmetric relation). A key result concerning the graphs $X^{p,q}$ is the following.

THEOREM 2.6. *The graph $X^{p,q}$ is a nonbipartite $(p+1)$-regular graph on $n=q(q^2-1)/2$ vertices. Moreover, if the eigenvalues of $X^{p,q}$ are $|\lambda_1|\geq\cdots\geq|\lambda_n|$, then $\lambda_1=p+1$ and*

$$|\lambda_j|\leq 2\sqrt{p}\quad \textit{for all}\quad j>1. \tag{2.8}$$

Because of (2.8), the graphs $X^{p,q}$ are called *Ramanujan graphs*. We now state the following well-known pseudorandom property of the graphs $X^{p,q}$, which follows from (2.8) (see, e.g., Corollary 9.2.5 in [5]).

COROLLARY 2.7. *The graph $X^{p,q}$ is $(\varrho,2)$-uniform, where $\varrho=(p+1)/n$.*

Having covered the basics of the Lubotzky, Phillips, and Sarnak construction, we turn to the proof of Lemma 2.5.

2.2.2. Proof of Lemma 2.5. We start with a simple lemma asserting the existence of appropriate primes p and q.

LEMMA 2.8. *There exists an absolute constant r_1 such that, for any $r_0\geq r_1$, there exists an integer $m_0=m_0(r_0)$ for which the following holds. There is an algorithm $\mathcal{P}$ that, on input (r_0,m), where $r_0\geq r_1$ and $m\geq m_0=m_0(r_0)$, produces a pair of primes p and q which satisfy*

$$p\neq q,\quad p,\,q\equiv 1\ (\mathrm{mod}\ 4),\quad \text{and}\quad \left(\frac{p}{q}\right)=1, \tag{2.9}$$

$$1.4r_0\leq p+1\leq 2r_0, \tag{2.10}$$

and

$$\sqrt[3]{2.1m}\leq q\leq 1.1\sqrt[3]{2.1m}. \tag{2.11}$$

Algorithm $\mathcal{P}$ runs in time $\leq C_1 m^{1/2}(\log m)^2$, where $C_1=C_1(r_0)$ depends only on r_0.

Proof. Let us start recalling Dirichlet's theorem on primes in arithmetic progressions. In particular, the quantitative version of Dirichlet's theorem implies that for integers a and b with $(a,b)=1$, there is an integer $t_{a,b}$ such that for all $t\geq t_{a,b}$ there is a prime $p\equiv a\ (\mathrm{mod}\ b)$ in the interval $[t,11t/10]=\{x\colon t\leq x\leq 11t/10\}$.

We let $r_1=t_{1,8}+1$ and proceed to show that this choice of r_1 will do. Thus, let an arbitrary integer $r_0\geq r_1$ be given, and let us define $m_0=m_0(r_0)$ as required in our lemma. To that end, first observe that, by the choice of r_1, there is a prime $p\equiv 1\ (\mathrm{mod}\ 8)$ satisfying (2.10). We fix such a prime p. Observe that we have $p\equiv 1\ (\mathrm{mod}\ 4)$. Moreover, since $p\equiv 1\ (\mathrm{mod}\ 8)$, we have that

$$2 \text{ is a quadratic residue modulo } p. \tag{2.12}$$

Since $(4,p)=1$, by the Chinese remainder theorem, there is a unique integer s with $1\leq s\leq 4p$ satisfying

$$s\equiv 2\pmod{p}\qquad\text{and}\qquad s\equiv 1\pmod{4}. \tag{2.13}$$

We are finally ready to define $m_0 = m_0(r_0)$. We let

$$m_0 = m_0(r_0) = \max\left\{\frac{10}{21}t_{s,4p}^3,\ \frac{8}{2.1}r_0^3\right\}. \tag{2.14}$$

Our aim now is to show that the choice for $m_0 = m_0(r_0)$ in (2.14) will do. Thus, let $m \geq m_0(r_0)$ be given. We shall now describe a procedure $\mathcal{P}$ to find the primes p and q as required. Our description will be quite informal.

The prime p with $p \equiv 1 \pmod 8$ satisfying (2.10) may be found easily. We now need to determine a suitable value for q. We choose q among the integers in the arithmetic progression $\{4pk + s\colon k = 0, 1, 2, \dots\}$, where s is the integer satisfying $1 \leq s \leq 4p$ and (2.13). By Dirichlet's theorem and our choice of $m_0 \geq (10/21)t_{s,4p}^3$, there is a prime $q \equiv s \pmod{4p}$ satisfying (2.11). We claim that our choice of s implies all properties promised for q. Indeed, $q \equiv s \equiv 1 \pmod 4$. Furthermore, the quadratic reciprocity law implies $(\frac{p}{q}) = (\frac{q}{p})$, since both $p, q \equiv 1 \pmod 4$. Using that $q \equiv s \equiv 2 \pmod p$ and recalling (2.12), we have

$$\left(\frac{p}{q}\right) = \left(\frac{q}{p}\right) = \left(\frac{s}{p}\right) = \left(\frac{2}{p}\right) = 1.$$

Finally, note that $m_0 \geq 8r_0^3/2.1$ guarantees $q > p$ and, consequently, condition (2.9) is satisfied. Therefore, the primes p and q, as required, do exist.

Let us now consider the time complexity of our procedure $\mathcal{P}$ above. We first observe that the search for $p < 2r_0$ takes a quantity of steps that depends only on r_0. To find q, we have enough time to check all integers in the interval $[\sqrt[3]{2.1m}, 1.1\sqrt[3]{2.1m}]$. Since this interval is of length $O(m^{1/3})$, this will take $O(m^{1/3} \cdot m^{1/6}(\log m)^2) = O(m^{1/2}(\log m)^2)$ steps. The $(\log m)^2$ term accounts for the time complexity of arithmetic operations with integers having $O(\log m)$ digits. □

We now describe algorithm $\mathcal{E}$, the existence of which is asserted in Lemma 2.5. Consider the integer r_1 and the function $m_0(r_0)$ whose existences are guaranteed by Lemma 2.8. On input (r_0, m), where $r_0 \geq r_1$ and $m \geq m_0(r_0)$, algorithm $\mathcal{E}$ performs the following steps.

1. Run algorithm $\mathcal{P}$ on input (r_0, m) to obtain primes p and q as in the statement of Lemma 2.8.
2. List all elements of $\mathrm{PSL}(2, \mathbb{Z}/q\mathbb{Z})$, i.e., the vertex set of $X^{p,q}$, by enumerating all the canonical representatives of the elements in $\mathrm{PSL}(2, \mathbb{Z}/q\mathbb{Z})$.
3. Find all solutions to $\alpha_0^2 + \alpha_1^2 + \alpha_2^2 + \alpha_3^2 = p$ that belong to S and construct T (see (2.7)).
4. For each vertex x of $X^{p,q}$, construct its adjacency list.
5. Set J to be any induced subgraph of $X^{p,q}$ on m vertices.

The following claim will finish the proof of Lemma 2.5.

Claim 2.9. *Algorithm $\mathcal{E}$ produces a graph J that is $(\varrho, 3)$-uniform in time $\leq Cm(\log m)^2$, where $r = \varrho m$ satisfies $r_0 \leq r \leq 2r_0$, and C is a constant that depends only on r_0.*

Proof. We start with the correctness of $\mathcal{E}$. We already know that $\mathcal{P}$ produces suitable primes p and q. Hence, we need only to argue that the graph J obtained in step 5 has the required properties.

By Theorem 2.6 and Corollary 2.7, the graph $X^{p,q}$ constructed in steps 2–4 has $n = q(q^2-1)/2$ vertices, is $(p+1)$-regular, and is $(\varrho, 2)$-uniform with $\varrho = (p+1)/n$. Furthermore, note that (2.11) implies $1 \leq n/m \leq 1.05(1.1)^3$. Lemma 2.1 implies

J is a (ϱ, L)-uniform graph with $\varrho = (p+1)/n$ and $L = 2\sqrt{n/m} \le 2.1\sqrt{231/200} = 2.256\cdots < 3$. Thus J is indeed a $(\varrho, 3)$-uniform graph.

Since $1 \le n/m \le 1.05(1.1)^3$, we deduce that $r = \varrho m = (p+1)m/n$ is such that

$$r \le 2r_0$$

and

$$r \ge 1.4r_0 \cdot \frac{20}{21}\left(\frac{10}{11}\right)^3 \ge r_0 \ .$$

Finally, we argue about the time complexity of each of the steps in algorithm $\mathcal{E}$. By Lemma 2.8, we already know that step 1 takes time $\le C_1(r_0)m(\log m)^2$, where $C_1(r_0)$ is a constant that depends only on r_0.

Recalling the form of the canonical representatives of the elements in $\mathrm{PSL}(2, \mathbb{Z}/q\mathbb{Z})$, we see that step 2 may be performed in time $O(m(\log m)^2)$. The time complexity of step 3 depends only on r_0.

In step 4, we take one by one the vertices of $X^{p,q}$, i.e., the elements of $\mathrm{PSL}(2, \mathbb{Z}/q\mathbb{Z})$, and generate their adjacency lists. Since $|T| = p+1 \le 2r_0$, to generate the adjacency list of a particular vertex takes only $O((\log q)^2)$ steps.

Finally, taking m vertices of $X^{p,q}$ arbitrarily and adjusting their adjacency lists to create an adjacency list representation for the corresponding induced subgraph takes $O(m \log m)$ time. Therefore, the time complexity of algorithm $\mathcal{E}$ is $\le C_0(r_0)m(\log m)^2$, as promised. □

3. Algorithms. Before we describe the algorithm, let us introduce some notation. Let $\Gamma = (V, E)$ be a graph and $v \in V$ a vertex. We write $\Gamma(v)$ for the neighborhood of v, i.e., for the set of all vertices adjacent to v in Γ, and $d(v)$ for the degree of v, i.e., $d(v) = |\Gamma(v)|$. To shorten our notation we will use that same letter to denote a graph and the set of its edges. For example, Γ will also stand for $E(\Gamma)$, and hence $e(\Gamma) = |\Gamma|$.

3.1. Regularity of bipartite graphs. In this section we describe algorithm $\mathcal{A}$ which takes as an input a bipartite graph $G = (A, B; E), |A| = |B| = m$, and $0 < \varepsilon < 1$. The algorithm in time $O(m^2)$ either confirms that G is ε-regular or finds sets $A' \subseteq A, B' \subseteq B, |A'| \ge \varepsilon' m, |B'| \ge \varepsilon' m$, such that

$$|d(A', B') - d(A, B)| \ge \varepsilon' \ .$$

Our algorithm $\mathcal{A}$ consists of the preprocessing stage $\mathcal{A}_P$ and the main procedure $\mathcal{A}_M$.

3.1.1. The preprocessing stage. In order to describe the preprocessing stage we need to define

$$\varepsilon' = \left(\frac{\varepsilon}{10}\right)^{20} \frac{1}{10^4} \ . \tag{3.1}$$

To describe $\mathcal{A}_P$ we need the constants ε and ε' only. Note that other constants are used for describing the other part $\mathcal{A}_M$; these other constants will be defined later in section 3.1.2 and will be related to ε and ε' above.

Algorithm $\mathcal{A}_P$ is based on the following standard observation and Lemma 3.1. We observe that if the bipartite graph G on vertex set $A \cup B, |A| = |B| = m$, is such that $p(G) := d(A, B) = |G|/m^2 \le \varepsilon^3$, then G is ε-regular (we omit the proof of this standard observation).

Lemma 3.1 quantifies a further observation that we may remove some vertices of our graph G so that we either obtain a subgraph $H \subseteq G$ that is essentially degree-regular (all degrees are about the same) or else in the process of removing these vertices we locate a witness to the ε'-irregularity of G. This is formalized as follows.

LEMMA 3.1. *Suppose G is a bipartite graph with vertex set $A \cup B$, $|A| = |B| = m$, and suppose that $p(G) > \varepsilon^3$ holds. There is a procedure that runs in time $O(m^2)$ that either* (i) *produces a witness to the ε'-irregularity of G or* (ii) *produces a bipartite subgraph $H \subseteq G$, say $H = (U, V; F)$, such that*

(a)

$$(1 - 2\varepsilon')m < |U|, |V| \leq m \,, \tag{3.2}$$

(b)

$$\big|p(H) - p(G)\big| \leq 5\varepsilon', \tag{3.3}$$

where $p(H) = |H|/|U||V|$, and

(c) *for all $u \in U$ and $v \in V$ we have*

$$\begin{aligned} d(u) &= (p(H) + O_1(10\varepsilon')) \cdot |V| \,, \\ d(v) &= (p(H) + O_1(10\varepsilon')) \cdot |U| \,. \end{aligned} \tag{3.4}$$

Proof. We first omit the vertices v in V for which the condition

$$d(v) = (p(G) + O_1(\varepsilon'))m \tag{3.5}$$

fails. If the number of such vertices is $\geq 2\varepsilon' m$, we may easily produce a witness to the ε'-irregularity of G as in (i) in the statement of our lemma. Suppose therefore that the number of such vertices is $< 2\varepsilon' m$. Let $V \subseteq B$ be the resulting subset of B. Hence $|V| > (1 - 2\varepsilon')m$. We now omit the vertices $u \in A$ for which the condition

$$d(u) = (p(G) + O_1(\varepsilon')) \cdot |V| \tag{3.6}$$

fails. Again, if the number of such vertices is $\geq 2\varepsilon' m$, we may easily produce a witness to the ε'-irregularity of G. If the number of such vertices is $< 2\varepsilon' m$, the resulting graph H is as in (ii) in the statement of our lemma.

The time complexity assertion will be verified in the proof of Lemma 3.4 (cf. algorithm $\mathcal{A}_P$ below). □

For convenience, we let $\Psi(m, \varepsilon')$ be the family of subgraphs H of G that satisfy (a)–(c) in (ii) of Lemma 3.1 above.

Now we are ready to describe *algorithm $\mathcal{A}_P$*:

1. Given G and ε, decide if $p(G) < \varepsilon^3$.
2. If $p(G) \leq \varepsilon^3$, then G is ε-regular and $\mathcal{A}_P$ halts.
3. If $p(G) > \varepsilon^3$, apply Lemma 3.1 to construct a subgraph H of G which satisfies (a)–(c). (Algorithm $\mathcal{A}_M$ will be applied to H.)

3.1.2. The main procedure. In view of step 3 in algorithm $\mathcal{A}_P$ (cf. Lemma 3.1) we will now assume that $H \in \Psi(m, \varepsilon')$.

(a) *Definition of constants.* Before describing algorithm $\mathcal{A}_M$ we will define constants needed for it. Recall that algorithm $\mathcal{A}$, and thus $\mathcal{A}_M$, is given $0 < \varepsilon < 1$. In section 3.1.1 we already defined $\varepsilon' = (\varepsilon/10)^{20}/10^4$.

We first let

$$\delta = \frac{1}{4}\Big(\frac{\varepsilon}{2}\Big)^5, \qquad L = 5, \quad \text{and} \quad r_A = \frac{10^6}{\varepsilon^8}\,. \tag{3.7}$$

We now let

$$\mu = \Big(\frac{\varepsilon}{10}\Big)^{10}\frac{1}{100} \tag{3.8}$$

and put

$$\eta = \frac{\mu}{3}\,. \tag{3.9}$$

We also let $r_B = \overline{r}(\eta, L)$ be as given in Lemma 2.4 and let

$$r_0 = \max\{r_A, r_B\}. \tag{3.10}$$

Finally, we set

$$\varepsilon_1 = \frac{1}{4}\left(\frac{\varepsilon}{2}\right)^{16}\,. \tag{3.11}$$

This will be used only later in a proof. However, it might be helpful to see the relation of ε_1 to the other constants introduced here.

The reader may find it useful to keep in mind the following hierarchy of the constants for ε small:

$$\varepsilon' < \varepsilon^{20} \ll \varepsilon_1 < \varepsilon^{16} \ll \eta = \frac{\mu}{3} < \varepsilon^{10} \ll \delta \ll \varepsilon^3 \ll \varepsilon,\, p\,. \tag{3.12}$$

(Here inequalities "$<$" are used to compare two quantities which differ by an absolute constant.) Note that for the description of $\mathcal{A}_M$ we need only to know r_0 and δ defined above. The other constants are needed in the proofs below.

(b) *Property* $\mathcal{P}(J,\delta)$. We introduce some notation. Let H be a bipartite graph with vertex set $U \cup V$. Let J be a (ϱ, L)-uniform graph with vertex set U. We say that H *has property* $\mathcal{P}(J,\delta)$ if

$$\sum_{\{u,u'\}\in J} \big|d_H(u,u') - p(H)^2\cdot|V|\big| \le \delta p(H)^2|V|\cdot|J| \tag{3.13}$$

holds. Recall that due to our notation $\{u,u'\} \in J$ means that $\{u,u'\}$ is an edge of J. Moreover, let us write J_v ($v \in V$) for the graph $J[H(v)]$ induced by the neighborhood $H(v)$ of v in H. We define a 0–1 matrix $M = (m(e,v))_{e,v}$ indexed by $J \times V$ as follows:

$$m(e,v) = \begin{cases} 1 & \text{if } e \in J_v,\\ 0 & \text{otherwise.}\end{cases} \tag{3.14}$$

Therefore, clearly, $m(e,v) = 1$ if and only if both endpoints of e are adjacent to v.

(c) *Description of* $\mathcal{A}_M$. We assume $\mathcal{A}_M$ is given a bipartite graph $H \in \Psi(m,\varepsilon')$ having vertex set $U \cup V$. *Algorithm* $\mathcal{A}_M$ now proceeds as follows:

1. Apply procedure $\mathcal{E}$ to construct a (ϱ, L)-uniform graph J with vertex set U and average degree $r = \varrho\cdot|U|$ satisfying

   $$r_0 = r_0(\eta) \le r \le 2r_0 \tag{3.15}$$

   (cf. Lemma 2.5).

2. Verify whether H has $\mathcal{P}(J,\delta)$. If it does, then G is ε-regular (see Lemma 3.2 below) and $\mathcal{A}_M$ halts.
3. If $\mathcal{P}(J,\delta)$ fails for H, construct matrix $M=(m(e,v))_{e,v}$ defined above. Find and fix a vertex $v_0\in V$ such that

$$\sum_{v\in V}\sum_{e\in J_{v_0}} m(e,v)\ge\left(1+\frac{\delta^2}{2}\right)p(H)^4|V|\cdot|J|. \tag{3.16}$$

(The existence of such a vertex v_0 is proved later; cf. Lemma 3.5.)
4. Set $U'=H(v_0)$ and

$$V'=\left\{v'\in V:\ \sum\{m(e,v'):\,e\in J_{v_0}\}\ge\left(1+\frac{\delta^2}{4}\right)p(H)^4\cdot|J|\right\}. \tag{3.17}$$

5. $\mathcal{A}_M$ outputs (U',V') and claims that this is a witness to the ε'-irregularity of G.

3.1.3. Correctness and analysis of algorithm $\mathcal{A}$. The correctness of $\mathcal{A}$ follows from Lemmas 3.2 and 3.3. The first lemma says that if algorithm $\mathcal{A}_M$, and hence $\mathcal{A}$, claims that G is ε-regular in step 2, then this is indeed the case.

LEMMA 3.2. *If H enjoys property $\mathcal{P}(J,\delta)$, then G is ε-regular.*

LEMMA 3.3. *If property $\mathcal{P}(J,\delta)$ fails for H, then G is not ε'-regular, and the pair (U',V') produced by algorithm $\mathcal{A}_M$ is indeed a witness for the ε'-irregularity of G.*

We shall prove the two lemmas above in section 4. The next two lemmas immediately imply that $\mathcal{A}$ has time complexity $O(m^2)$.

LEMMA 3.4. *Algorithm $\mathcal{A}_P$ described in section* 3.1.1 *has time complexity $O(m^2)$.*

Proof. The steps of $\mathcal{A}_P$ have the following time complexity. The only computations are in steps 1 and 3.

Step 1. Since $p(G)=|G|/m^2$, it takes at most $O(m^2)$ steps to compute $p(G)$ and decide whether $p(G)<\varepsilon^3$.

Step 3. Based on the proof of Lemma 3.1, a vertex of G is put into H if and only if it satisfies (3.4). Hence, we proceed through all $2m$ vertices of G, each time checking (3.4), which takes $O(m)$ steps. Thus, this step takes $O(m^2)$ steps, too.

The overall time complexity of $\mathcal{A}_P$ is $O(m^2)$. □

LEMMA 3.5. *Algorithm $\mathcal{A}_M$ described in section* 3.1.2 *runs in time $O(m^2)$.*

Proof. The steps of $\mathcal{A}_M$ have the following time complexity.

Step 1. To perform procedure $\mathcal{E}$ we need $O(m(\log m)^2)$ steps; cf. Lemma 2.5.

Step 2. To verify $\mathcal{P}(J,\delta)$ we need to add $|J|=O(m)$ summands. Computing each of these summands takes $O(m)$ steps. Consequently, one can decide $\mathcal{P}(J,\delta)$ in $O(m^2)$ time.

Step 3. Deciding if $m(e,v)=1$ or 0 can be performed in constant time. Thus, constructing M takes $O(|J||V|)=O(m^2)$ time. To check (3.16), we first write it in the equivalent form

$$\sum_{e\in J_{v_0}}\sum_{v\in V} m(e,v)\ge\left(1+\frac{\delta^2}{2}\right)p(H)^4|V|\cdot|J|\,. \tag{3.18}$$

Using matrix M we compute the column sums $\sum_{v\in V}m(e,v)$ for each $e\in J$. This takes $O(m^2)$ steps.

Now to check (3.18) for a fixed vertex $v_0 \in V$, we first find J_{v_0}, which takes $|J_{v_0}| = O(m)$ steps. Then we add together the column sums $\sum_{v\in V} m(e,v)$ for all $e \in J_{v_0}$ and decide about the truth of (3.18). This takes another $O(m)$ steps. In the worst case we need to check all $v_0 \in V$. Since $|V| = m$, to find v_0 that satisfies (3.18) will take at most $O(m^2)$ steps.

Step 4. Since for each $v' \in V$ the condition $\sum_{e\in J_{v_0}} m(e,v') \geq (1+\delta^2/4)p(H)^4|J|$ can be verified in $O(m)$ time, the set V' can be constructed in $O(m^2)$ time.

Step 5. This step takes a constant time.

Therefore, the time complexity of $\mathcal{A}_M$ is $O(m^2)$. Finally, let us point out that the fact that J has $O(m)$ edges was crucial. □

3.2. The regularity lemma. For the sake of completeness, we include an algorithm necessary for proving Corollary 1.6. Given Theorem 1.5 we can derive the necessary algorithm in a standard way.

Let $V_0, V_1, \ldots, V_k$ be an equitable partition P of the set of vertices of a graph. We define the *index* of P (cf. [31]) by

$$\operatorname{ind}(P) = \frac{1}{k^2} \sum_{1\leq r<s\leq k} d(V_r, V_s)^2 .$$

To present a proof of Corollary 1.6 we will use the following lemma, which was proved in [31]. Note that no comment is made in [31] on the running time. However, the proof of the lemma implies an algorithm of time complexity $O(n)$.

LEMMA 3.6. *Fix k and γ, and let $G = (V,E)$ be a graph on n vertices. Let P be an equitable partition of V into classes $V_0, V_1, \ldots, V_k$. Assume $|V_1| > 4^{2k}$ and $4^k > 600\gamma^{-5}$. Given proofs that more than γk^2 pairs (V_r, V_s) are not γ-regular (where by proofs we mean subsets $X = X(r,s) \subseteq V_r$, $Y = Y(r,s) \subseteq V_s$ that violate the condition of γ-regularity of (V_r, V_s)), then one can find in $O(n)$ time an equitable partition P' (which is a refinement of P) into $1 + k4^k$ classes, with the exceptional class of cardinality at most $|V_0| + n/4^k$, and such that*

$$\operatorname{ind}(P') \geq \operatorname{ind}(P) + \frac{\gamma^5}{20} .$$

Proof of Corollary 1.6. Theorem 1.5 and Lemma 3.6 already imply Corollary 1.6. Let $\varepsilon > 0$ and k_0 be a positive integer. Let $\varepsilon' = \varepsilon^{20}/10^{24}$. We set $N = N(\varepsilon, k_0)$ and $T = T(\varepsilon, k_0)$ as follows: Let a be the least positive integer such that

$$4^a > 600 \left(\frac{\varepsilon'}{4}\right)^{-5}, \quad a \geq k_0 . \tag{3.19}$$

Let k_i be a sequence of integers defined inductively as

$$k_0 = a, \quad k_{i+1} = k_i 4^{k_i} .$$

Set $T = k_{\lceil 10(\varepsilon'/4)^{-5}\rceil}$ and $N = \max\{T4^{2T}, 2T/{\varepsilon'}^2\}$. Finally, we set $K_0'' = N \geq T$.

Let $\Gamma = (V,E)$ be a graph on n vertices, $n \geq N$. The following algorithm constructs an ε-regular partition of Γ into $k+1$ classes with $k_0 \leq k \leq T \leq K_0''$.

Algorithm $\mathcal{A}_0'$ proceeds as follows:

1. Arbitrarily divide the vertices of Γ into an equitable partition P_1 with classes $V_0, V_1, \ldots, V_a$, where $|V_1| = \lfloor n/a \rfloor$ and $|V_0| < a$. Set $k_1 = a$.

2. For every pair (V_r, V_s) of P_i, verify if it is ε-regular or find $V_r' \subseteq V_r$, $V_s' \subseteq V_s$, $|V_r'| \geq (\varepsilon'/4)|V_r|$, $|V_s'| \geq (\varepsilon'/4)|V_s|$, such that $|d(V_r', V_s') - d(V_r, V_s)| \geq \varepsilon'$.
3. If there are at most $\varepsilon\binom{k_i}{2}$ pairs that are not verified as ε-regular, then stop. The partition P_i is an ε-regular partition.
4. Apply Lemma 3.6, where $P = P_i$, $k = k_i$, and $\gamma = \varepsilon'/4$, and obtain a partition P' with $1 + k_i 4^{k_i}$ classes.
5. Let $k_{i+1} = k_i 4^{k_i}$, $P_{i+1} = P'$, and $i = i + 1$, and go to step 2.

CLAIM 3.7. *Algorithm $\mathcal{A}_0'$ as described is correct and runs in $O(n^2)$ time.*

Proof. To prove correctness of algorithm $\mathcal{A}_0'$ is quite standard. Since the index of partitions P_i constructed by $\mathcal{A}_0'$ strictly increases and at the same time is bounded by 1 from above, algorithm $\mathcal{A}_0'$ uses Lemma 3.6, and thus Theorem 1.5, only finitely many times. Each such use takes $O(n^2)$ time. □

4. Proofs of the main lemmas. We use the notation introduced in section 3.1. Before the proofs let us point out a straightforward estimate on the size of neighborhoods used throughout the proofs.

Remark 4.1. Let $H \in \Psi(m, \varepsilon')$ be the graph in the statement of Lemma 3.1, and let $u \in U$ and $v \in V$ be vertices of H. As an immediate consequence of (3.4) and the definition of ε' and μ, we get

$$\begin{aligned} |H(u)| &= (p(H) + O_1(10\varepsilon'))|V| = (1 + O_1(\mu))p(H)|V| \,, \\ |H(v)| &= (p(H) + O_1(10\varepsilon'))|U| = (1 + O_1(\mu))p(H)|U| \,. \end{aligned} \tag{4.1}$$

4.1. Proof of Lemma 3.2. Let $H \in \Psi(m, \varepsilon')$ be the input graph of algorithm $\mathcal{A}_M$. Let $U \cup V$ be the vertex set of H. Set $m_U = |U|$, $m_V = |V|$, and $p := p(H) = |H|/m_U m_V$.

We say that the graph H has *property* $\mathcal{Q}(J, \delta)$ if the inequality

$$\sum_{\{u,u'\} \in J} \left| d_H(u, u') + p^2 m_V - (d_H(u) + d_H(u'))p \right| \leq \delta p^2 m_V \cdot |J| \tag{4.2}$$

holds true.

CLAIM 4.2. *Let $\delta > 0$ be fixed. If a graph $H \in \Psi(m, \varepsilon')$ has property $\mathcal{P}(J, \delta)$, then it has property $\mathcal{Q}(J, 2\delta)$.*

Proof. Since H enjoys $\mathcal{P}(J, \delta)$ and $H \in \Psi(m, \varepsilon')$, we have

$$\begin{aligned} &\sum_{\{u,u'\} \in J} \left| d_H(u, u') + p^2 m_V - (d_H(u) + d_H(u'))p \right| \\ &\qquad \leq \sum_{\{u,u'\} \in J} \left| d_H(u, u') - p^2 m_V \right| \\ &\qquad\qquad + 2p \sum_{u \in U} \left| p m_V - d_H(u) \right| d_J(u) \\ &\qquad \leq \delta p^2 m_V |J| + 20 p \varepsilon' m_V \sum_{u \in U} d_J(u) \\ &\qquad \leq 2\delta p^2 m_V |J| \,, \end{aligned}$$

since $\varepsilon' \leq \delta\varepsilon^3/40 \leq \delta p/40$ (see (3.1), (3.7)). □

In view of Claim 4.2, we are going to prove the following claim.

CLAIM 4.3. *Every $H \in \Psi(m, \varepsilon')$ that has property $\mathcal{Q}(J, 2\delta)$ is $(\varepsilon/2)$-regular.*

The proof is presented below. First, let us observe that the $\varepsilon/2$-regularity of H implies the ε-regularity of G. Indeed, let $X \subseteq A$, $|X| \geq \varepsilon|A|$, and $Y \subseteq B$, $|Y| \geq \varepsilon|B|$. Set $X' = X \cap U$ and $Y' = Y \cap V$. By Lemma 3.1,

$$|X'| \geq \varepsilon|A| - 2\varepsilon'|A| \geq (\varepsilon - 2\varepsilon')|U| \geq \frac{\varepsilon}{2}|U|$$

and, similarly,

$$|Y'| \geq \frac{\varepsilon}{2}|V| \,.$$

A standard argument based on the fact that X' and Y' are almost equal to X and Y (recall Lemma 3.1) shows that $|d(X,Y) - d(X',Y')| \leq \varepsilon/4$. Thus, using the $\varepsilon/2$-regularity of H and Lemma 3.1, we get

$$\begin{aligned}|d(X,Y) - d(A,B)| &\leq |d(X,Y) - d(X',Y')| + |d(X',Y') - d(U,V)| \\ &\quad + |d(U,V) - d(A,B)| \leq \frac{\varepsilon}{4} + \frac{\varepsilon}{2} + 5\varepsilon' < \varepsilon \,.\end{aligned}$$

Hence, G is ε-regular. □

Proof of Claim 4.3. Let $\mathbb{A} = (a_{u,v})_{u,v}$ be a matrix indexed by $U \times V$ with entries

$$a_{u,v} = \begin{cases} -(1-p) & \text{if } \{u,v\} \in H, \\ p & \text{otherwise.} \end{cases}$$

Moreover, for $u \in U$, let $\xi_u = (a_{u,1}, \ldots, a_{u,m_V})$ be the uth row of $\mathbb{A}$. The following claim follows easily from the definition of property $\mathcal{Q}$ (see (4.2)).

Claim 4.4. *For every $H \in \Psi(m,\varepsilon')$ that has property $\mathcal{Q}(J,2\delta)$, the row-vectors of $\mathbb{A}$ satisfy the following inequality:*

$$\sum_{\{u,u'\} \in J} |\langle \xi_u, \xi_{u'} \rangle| \leq 2\delta p^2 m_V |J| \,. \tag{4.3}$$

Proof. Since

$$\begin{aligned}\langle \xi_u, \xi_{u'} \rangle &= d_H(u,u')(1-p)^2 - (d_H(u) + d_H(u') - 2d_H(u,u'))p(1-p) \\ &\quad + (m_V - (d_H(u) + d_H(u') - d_H(u,u'))p^2 \\ &= d_H(u,u')[(1-p)^2 + 2p(1-p) + p^2] + m_V p^2 - (d_H(u) + d_H(u'))p \\ &= d_H(u,u') + m_V p^2 - (d_H(u) + d_H(u'))p\end{aligned}$$

for any pair of vertices $u, u' \in U$, we have that $\sum_{\{u,u'\} \in J} |\langle \xi_u, \xi_{u'} \rangle|$ equals the sum on the left-hand side of (4.2). Thus the claim follows. □

Let $U' \subseteq U$ and $V' \subseteq V$. To shorten our notation $\sum_{u,u' \in J}^{U'}$ will denote summation over $\{u,u'\} \in J$ such that $u, u' \in U'$. Furthermore, for $u \in U'$ let ψ_u be the restriction of the vector ξ_u to V', i.e., $\psi_u = (a_{u,v})_{v \in V'}$. We clearly have $\sum_{u,u' \in J}^{U'} |\langle \xi_u, \xi_{u'} \rangle| \leq \sum_{\{u,u'\} \in J} |\langle \xi_u, \xi_{u'} \rangle|$. We now compare $\sum_{u,u' \in J}^{U'} \langle \xi_u, \xi_{u'} \rangle$ with $\sum_{u,u' \in J}^{U'} \langle \psi_u, \psi_{u'} \rangle$. We have

$$\sum\nolimits_{u,u' \in J}^{U'} \langle \xi_u, \xi_{u'} \rangle = \sum\nolimits_{u,u' \in J}^{U'} \langle \psi_u, \psi_{u'} \rangle + \sum_{v \notin V'} \sum\nolimits_{u,u' \in J}^{U'} a_{u,v} a_{u',v}. \tag{4.4}$$

For $v \in V$ we set $S_v^{U'} = \sum_{u,u' \in J}^{U'} a_{u,v} a_{u',v}$ and proceed to estimate this quantity. Let $S := H(v) \cap U' \subseteq U'$ be the set of neighbors of the vertex v in U' and $T := (U' \setminus S) \subseteq U'$ be the set of nonneighbors of the vertex v in U'.

Set $\alpha = |S|/|U'|$ and $\beta = |T|/|U'|$. Note $\alpha + \beta = 1$. Thus we can write

$$S_v^{U'} = e_J(S)(1-p)^2 + e_J(T)p^2 - e_J(S,T)p(1-p). \tag{4.5}$$

The (ϱ, L)-uniformity of J implies the following claim. Recall that for two numbers a, b we write $a = O_1(b)$ if and only if $|a| \le b$.

CLAIM 4.5. *For all $v \in V'$ and $U' \subseteq U$, we have*

$$S_v^{U'} = \frac{\varrho}{2}|U'|^2 \cdot \big[\alpha(1-p) - \beta p\big]^2 + O_1\big(3L\sqrt{r} \cdot |U'|\big)\,. \tag{4.6}$$

Proof. Set $m' = |U'|$. Since J is a (ϱ, L)-uniform graph, Lemma 2.3 implies

$$\begin{aligned}
e_J(S) &= \frac{\varrho}{2}|S|^2 + O_1\big(2L\sqrt{r} \cdot |S|\big) \\
&= \frac{\varrho}{2}(\alpha m')^2 + O_1\big(2L\sqrt{r} \cdot \alpha m'\big)\,, \\
e_J(T) &= \frac{\varrho}{2}|T|^2 + O_1\big(2L\sqrt{r} \cdot |T|\big) \\
&= \frac{\varrho}{2}(\beta m')^2 + O_1\big(2L\sqrt{r} \cdot \beta m'\big)\,, \\
e_J(S,T) &= \varrho|S| \cdot |T| + O_1\big(L\sqrt{r|S| \cdot |T|}\big) \\
&= \varrho\alpha\beta(m')^2 + O_1\big(L\sqrt{r\alpha\beta} \cdot m'\big)\,.
\end{aligned}$$

Using (4.5) we get

$$\begin{aligned}
S_v^{U'} &= \Big[\frac{\varrho}{2}(\alpha m')^2 + O_1\big(2L\sqrt{r} \cdot \alpha m'\big)\Big](1-p)^2 \\
&\qquad + \Big[\frac{\varrho}{2}(\beta m')^2 + O_1\big(2L\sqrt{r} \cdot \beta m'\big)\Big]p^2 \\
&\qquad - \big[\varrho\alpha\beta(m')^2 + O_1\big(L\sqrt{r\alpha\beta} \cdot m'\big)\big]p(1-p) \qquad (4.7) \\
&= \frac{\varrho}{2}(m')^2\big[\alpha(1-p) - \beta p\big]^2 + \Delta,
\end{aligned}$$

where

$$\Delta = O_1\big(2L\sqrt{r} \cdot \alpha m'(1-p)^2 + 2L\sqrt{r} \cdot \beta m' p^2 + L\sqrt{r\alpha\beta} \cdot m' p(1-p)\big)\,. \tag{4.8}$$

To bound Δ we are going to use the inequalities $\alpha(1-p)^2 + \beta p^2 \le \alpha + \beta$ and $\sqrt{\alpha\beta} \le (\alpha + \beta)$. Thus,

$$\begin{aligned}
|\Delta| &\le 2L\sqrt{r} \cdot \alpha m'(1-p)^2 + 2L\sqrt{r} \cdot \beta m' p^2 + L\sqrt{r\alpha\beta} \cdot m' p(1-p) \\
&\le L\sqrt{r} m'\big(2\alpha + 2\beta + \sqrt{\alpha\beta}\big) \\
&\le L\sqrt{r} m' \cdot 3(\alpha + \beta) \qquad (4.9) \\
&= 3L\sqrt{r} m'\,.
\end{aligned}$$

Expressions (4.7) and (4.9) already imply the claim. We note only that we got a bound on Δ linear in $m' = |U'|$ since L and r are constants. ∎

Next we proceed with an upper and lower bound on $\sum_{u,u'\in J}^{U'}\langle\psi_u,\psi_{u'}\rangle$. To derive an upper bound for this quantity we use (4.4) to relate $\sum_{u,u'\in J}^{U'}\langle\psi_u,\psi_{u'}\rangle$ and $S_v^{U'}$ as follows:

$$\sum\nolimits_{u,u'\in J}^{U'}\langle\psi_u,\psi_{u'}\rangle=\sum\nolimits_{u,u'\in J}^{U'}\langle\xi_u,\xi_{u'}\rangle-\sum_{v\notin V'}S_v^{U'}.$$

Note that (4.6) implies $S_v^{U'}\ge -3L\sqrt{r}|U'|$. This lower bound and Claim 4.4 imply

$$\sum\nolimits_{u,u'\in J}^{U'}\langle\psi_u,\psi_{u'}\rangle\le 2\delta p^2 m_V|J|+3L\sqrt{r}|U'|(m_V-|V'|)\,. \tag{4.10}$$

To estimate $\sum_{u,u'\in J}^{U'}\langle\psi_u,\psi_{u'}\rangle$ from below, we first write

$$\sum\nolimits_{u,u'\in J}^{U'}\langle\psi_u,\psi_{u'}\rangle=\sum_{v\in V'}\sum\nolimits_{u,u'\in J}^{U'}a_{u,v}a_{u',v}=\sum_{v\in V'}S_v^{U'}\,. \tag{4.11}$$

Using (4.6) we get a lower bound on the expression in our last equation:

$$\begin{aligned}\sum\nolimits_{u,u'\in J}^{U'}\langle\psi_u,\psi_{u'}\rangle&=\sum_{v\in V'}S_v^{U'}\\ &\ge\sum_{v\in V'}\Big(\frac{\varrho}{2}|U'|^2\big[\alpha(1-p)-\beta p\big]^2-3L\sqrt{r}\cdot|U'|\Big)\\ &\ge\frac{\varrho}{2}\cdot\frac{|U'|^2}{|V'|}\left[\sum_{v\in V'}(\alpha(1-p)-\beta p)\right]^2-3L\sqrt{r}|U'|\cdot|V'|\,.\end{aligned} \tag{4.12}$$

We used the Cauchy–Schwarz inequality to get the last line of our bound. Comparing the lower and upper bounds on $\sum_{u,u'\in J}^{U'}\langle\psi_u,\psi_{u'}\rangle$ (cf. (4.12) and (4.10)), we infer

$$\Big[\sum_{v\in V'}(\alpha(1-p)-\beta p)\Big]^2\le\frac{2|V'|}{\varrho|U'|^2}\cdot\big(2\delta p^2m_V|J|+3L\sqrt{r}|U'|m_V\big)\,. \tag{4.13}$$

Now we are ready to show that H is $\frac{\varepsilon}{2}$-regular. Fix $U'\subseteq U$, $|U'|\ge\frac{\varepsilon}{2}|U|=\frac{\varepsilon}{2}m_U$, and $V'\subseteq V$, $|V'|\ge\frac{\varepsilon}{2}|V|=\frac{\varepsilon}{2}m_V$. Recall that $d(U,V)=p=p(H)$ in our notation. First we relate the difference of densities to the left-hand side of (4.13) as follows:

$$\begin{aligned}|d(U',V')-d(U,V)|^2&=\left|\frac{e(U',V')}{|U'||V'|}-p\right|^2\\ &=\frac{1}{|U'|^2|V'|^2}\Big[e(U',V')-p|U'|\cdot|V'|\Big]^2\\ &=\frac{1}{|U'|^2|V'|^2}\Big[\sum_{v\in V'}(|H(v)\cap U'|-p|U'|)\Big]^2\\ &=\frac{1}{|U'|^2|V'|^2}\Big[\sum_{v\in V'}\alpha|U'|-p|U'|\Big]^2\\ &=\frac{1}{|V'|^2}\Big[\sum_{v\in V'}(\alpha(1-p)-\beta p)\Big]^2\,.\end{aligned} \tag{4.14}$$

Finally, we bound the expression for the difference of densities in (4.14) using (4.13). We get

$$\begin{aligned}\big|d(U',V')-d(U,V)\big|^2 &\le \frac{2}{\varrho|U'|^2|V'|}\left(2\delta p^2 m_V|J| \ + 3L\sqrt{r}\cdot|U'|m_V\right)\\ &\le \frac{2}{\varrho|U'|^2|V'|}\times 2\delta p^2 m_V\times\frac{rm_U}{2}+\frac{6L\sqrt{r}\cdot|U'|m_V}{\varrho|U'|^2|V'|}\\ &\le \frac{2\delta p^2 {m_U}^2 m_V}{|U'|^2|V'|}+\frac{6Lm_Vm_U\sqrt{r}}{r|U'|\cdot|V'|}\\ &\le \frac{16\delta}{\varepsilon^3}+\frac{24L}{\sqrt{r}\cdot\varepsilon^2}\\ &\le \frac{1}{2}\cdot\left(\frac{\varepsilon}{2}\right)^2+\frac{1}{2}\cdot\left(\frac{\varepsilon}{2}\right)^2\\ &= \left(\frac{\varepsilon}{2}\right)^2.\end{aligned}$$

The last inequality follows because of our choice of δ and $r\ge r_0\ge r_A$. □

4.2. Proof of Lemma 3.3. For this proof, the reader may find it convenient to recall the hierarchy of the constants given in (3.12). Recall that we have

$$p=p(H)=p(G)+O_1(5\varepsilon')\ge\frac{1}{2}\varepsilon^3. \tag{4.15}$$

Note that (4.15) is guaranteed to hold for the graph H that we obtain after preprocessing H as described in section 3.1.1 since $p(G)\ge\varepsilon^3$ in this case. As in section 3.1.2 we assume $U\cup V$ is the vertex set of H and set $m_U=|U|$ and $m_V=|V|$.

Suppose that property $\mathcal{P}(J,\delta)$ fails for H. Thus

$$\sum_{\{u,u'\}\in J}|d_H(u,u')-p^2m_V|>\delta p^2m_V|J|\,. \tag{4.16}$$

Let us first observe that

$$\begin{aligned}&\sum_{\{u,u'\}\in J}\big(d_H(u,u')-p^2m_V\big)^2\\ &\quad=\sum_{\{u,u'\}\in J}\big(d_H(u,u')^2-2d_H(u,u')p^2m_V+p^4{m_V}^2\big) \qquad (4.17)\\ &\quad=\sum_{\{u,u'\}\in J}d_H(u,u')^2-2p^2m_V\sum_{\{u,u'\}\in J}d_H(u,u')+p^4{m_V}^2|J|\,.\end{aligned}$$

However,

$$\begin{aligned}\sum_{\{u,u'\}\in J}d_H(u,u')&=\sum_{\{u,u'\}\in J}|H(u)\cap H(u')|\\ &=\sum_{v\in V}|J_v|=\sum_{v\in V}(1+O_1(\eta))(1+O_1(\mu))^2p^2|J|\,.\end{aligned}$$

In the last inequality we used that due to preprocessing (cf. (4.1)), $|H(v)|=(1+O_1(\mu))pm_U$, and hence (a) in Lemma 2.4 gives (note that $(1+O_1(\mu))p\ge\eta$)

$$|J_v|=(1+O_1(\eta))(1+O_1(\mu))^2p^2|J|=(1+O_1(3\mu))p^2|J|\,.$$

Therefore

$$\sum_{\{u,u'\}\in J} d_H(u,u') = (1+O_1(3\mu))p^2 m_V |J| \,. \tag{4.18}$$

From (4.17) and (4.18) we obtain that

$$\begin{aligned}\sum_{\{u,u'\}\in J} \big(d_H(u,u') - p^2 m_V\big)^2 &\\ = \sum_{\{u,u'\}\in J} d_H(u,u')^2 - (1+O_1(6\mu))p^4 {m_V}^2 |J| &\\ \le \sum_{\{u,u'\}\in J} d_H(u,u')^2 - \left(1-\frac{\delta^2}{2}\right) p^4 {m_V}^2 |J| \,. &\end{aligned} \tag{4.19}$$

The last inequality holds true due to our choices of δ and μ; cf. (3.7), (3.8). On the other hand, in view of (4.16), we have by the Cauchy–Schwarz inequality that

$$\begin{aligned}\sum_{\{u,u'\}\in J} \big(d_H(u,u') - p^2 m_V\big)^2 &\ge \frac{1}{|J|}\Big(\sum_{\{u,u'\}\in J} |d_H(u,u') - p^2 m_V|\Big)^2 \\ &> \frac{1}{|J|}\big(\delta |J| p^2 m_V\big)^2 = \delta^2 p^4 {m_V}^2 |J| \,.\end{aligned} \tag{4.20}$$

Comparing (4.19) and (4.20), we deduce that

$$\sum_{\{u,u'\}\in J} d_H(u,u')^2 \ge \left(1+\frac{\delta^2}{2}\right) p^4 {m_V}^2 |J| \,. \tag{4.21}$$

We shall now evaluate the sum on the left-hand side of (4.21) in terms of the matrix $M = (m(e,v))_{e,v}$ defined in section 3.1.

Clearly, if $e = \{u,u'\} \in J$, then

$$d_H(u,u') = |H(u) \cap H(u')| = \sum_{v\in V} m(e,v),$$

and hence

$$d_H(u,u')^2 = \sum_{v\in V}\sum_{v'\in V} m(e,v) m(e,v') \,.$$

Therefore

$$\begin{aligned}\sum_{\{u,u'\}\in J} d_H(u,u')^2 &= \sum_{v\in V}\sum_{e\in J} m(e,v) \sum_{v'\in V} m(e,v') \\ &= \sum_{v\in V}\sum_{e\in J_v}\sum_{v'\in V} m(e,v') = \sum_{v\in V}\sum_{v'\in V}\sum_{e\in J_v} m(e,v') \,.\end{aligned} \tag{4.22}$$

Comparing (4.21) and (4.22) we infer that

$$\sum_{v\in V}\sum_{v'\in V}\sum_{e\in J_v} m(e,v') \ge \left(1+\frac{\delta^2}{2}\right) p^4 {m_V}^2 |J|,$$

and hence there is a vertex $v_0 \in V$ for which we have

$$\sum_{v' \in V} \sum_{e \in J_{v_0}} m(e, v') \geq \left(1 + \frac{\delta^2}{2}\right) p^4 m_V |J| \,. \tag{4.23}$$

Following the algorithm, we fix such a vertex v_0. We now *set*

$$V' := \left\{ v' \in V \,:\, \sum_{e \in J_{v_0}} m(e, v') \geq \left(1 + \frac{\delta^2}{4}\right) p^4 |J| \right\} . \tag{4.24}$$

As in the algorithm, we put $U' = H(v_0)$. One may prove that both U' and V' are large sets. The proof is postponed for the next section.

CLAIM 4.6. *$|U'| \geq \varepsilon_1 m_U$ and $|V'| \geq \varepsilon_1 m_V$.*

Recall that we defined $\varepsilon_1 = \frac{1}{4}\left(\frac{\varepsilon}{2}\right)^{16}$ in (3.11) to satisfy $\varepsilon' \ll \varepsilon_1 \ll \delta^2$. Before we proceed, using the definition of matrix M we observe that for all $v' \in V$ we have

$$\sum_{e \in J_{v_0}} m(e, v') = e(J[H(v_0) \cap H(v')]) \,. \tag{4.25}$$

Combining (4.25) and the fact that the edges of J are extremely well distributed we shall now provide a lower bound on $d_H(v_0, v')$, where v_0 is the vertex fixed above and v' is an arbitrary vertex of V'.

CLAIM 4.7. *For all $v' \in V'$, we have*

$$d_H(v_0, v') \geq \left(1 + \frac{\delta^2}{12}\right) p^2 m_U \,. \tag{4.26}$$

The proof of Claim 4.7 is given in next section. Since $U' = H(v_0)$, it follows immediately from Claim 4.7 that

$$e_H(U', V') \geq \left(1 + \frac{\delta^2}{12}\right) p^2 m_U |V'|, \tag{4.27}$$

which implies that

$$\begin{aligned} d_H(U', V') = \frac{e_H(U', V')}{|U'||V'|} &\geq \frac{(1 + \delta^2/12) p^2 m_U |V'|}{(1+\mu) p m_U |V'|} \\ &\geq \left(1 + \frac{\delta^2}{14}\right) p > p + \varepsilon_1 \,. \end{aligned} \tag{4.28}$$

Since we have already proved that $|U'| \geq \varepsilon_1 m_U$ and $|V'| \geq \varepsilon_1 m_V$ (see Claim 4.6), inequality (4.28) tells us that (U', V') is a witness to the ε_1-irregularity of H.

We shall now prove that (U', V') is in fact a witness to the ε'-irregularity of G. We have

$$|U'| \geq \varepsilon_1 m_U \geq \varepsilon_1 (1 - 2\varepsilon') m \geq \varepsilon' m \tag{4.29}$$

and, similarly,

$$|V'| \geq \varepsilon_1 m_V \geq \varepsilon_1 (1 - 2\varepsilon') m \geq \varepsilon' m \,. \tag{4.30}$$

Because of (3.3) and (4.28), we have

$$d(U', V') > p + \varepsilon_1 \geq p(G) - 5\varepsilon' + \varepsilon_1 \geq p(G) + \varepsilon'. \tag{4.31}$$

In view of (4.29), (4.30) inequality (4.31) implies that (U', V') is indeed a witness to the ε'-irregularity of G, as required.

4.2.1. Proofs of Claims 4.6 and 4.7. Here we give proofs of Claims 4.6 and 4.7.

Proof of Claim 4.6. Since $U' = H(v_0)$, estimates (4.1) and our definition of ε_1 imply $|U'| = (1 + O_1(\mu))pm_U \geq \varepsilon_1 m_U$.

Now we will give a lower bound on $|V'|$. By the definition of V', we have

$$\begin{aligned}\sum_{v'\in V}\sum_{e\in J_{v_0}} m(e,v') &= \sum_{v'\notin V'}\sum_{e\in J_{v_0}} m(e,v') + \sum_{v'\in V'}\sum_{e\in J_{v_0}} m(e,v') \\ &< \left(1+\frac{\delta^2}{4}\right)p^4|J|(m_V - |V'|) + |V'|e(J_{v'})\ . \end{aligned} \tag{4.32}$$

Since $|H(v')| = (1 + O_1(\mu))pm_V$ (cf. (4.1)), (a) in Lemma 2.4 implies (note $(1 + O_1(\mu))p \geq \eta$)

$$e(J_{v'}) = e(J[H(v')]) = (1 + O_1(\eta))(1 + O_1(\mu))^2p^2|J| = (1 + O_1(3\mu))p^2|J|\ .$$

Thus continuing with (4.32) we can write

$$\sum_{v'\in V}\sum_{e\in J_{v_0}} m(e,v') < \left(1+\frac{\delta^2}{4}\right)p^4|J|m_V + (1 + O_1(3\mu))|V'|p^2|J|\ . \tag{4.33}$$

Comparing (4.23) and (4.33), we obtain

$$2|V'| \geq (1 + O_1(3\mu))|V'| \geq \frac{1}{4}\delta^2p^2m_V,$$

and this, using the definition of δ and ε_1, gives

$$|V'| \geq \frac{1}{8}\delta^2p^2m_V \geq \frac{1}{128}\left(\frac{\varepsilon}{2}\right)^{10}\varepsilon^6m_V \geq \varepsilon_1 m_V\ ,$$

as required. □

Proof of Claim 4.7. Suppose to the contrary that (4.26) fails, i.e., $d_H(v_0, v') < (1 + \delta^2/12)p^2m_U$. We distinguish two cases: If $d_H(v_0, v') \geq \eta m_U$, then using (a) in Lemma 2.4 for $H(v_0) \cap H(v')$ implies

$$e(J[H(v_0) \cap H(v')]) < (1 + O_1(\eta))\left(1+\frac{\delta^2}{12}\right)^2 p^4|J|\ .$$

If, on the other hand, $d(v_0, v') < \eta m_U$ we have (cf. (b) in Lemma 2.4)

$$e(J[H(v_0) \cap H(v')]) < 2\eta^2|J|\ .$$

In either case, we have

$$e(J[H(v_0) \cap H(v')]) < \left(1+\frac{\delta^2}{4}\right)p^4|J|\ . \tag{4.34}$$

However, in view of the definition of V' (see (4.24), (4.25)), inequality (4.34) cannot hold. This contradiction shows that (4.26) must indeed hold. □

Acknowledgment. The authors are grateful to the referee for his or her detailed comments.

REFERENCES

[1] N. Alon, R. A. Duke, H. Lefmann, V. Rödl, and R. Yuster, *The algorithmic aspects of the regularity lemma (extended abstract)*, in Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, PA, 1992, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 473–481.

[2] N. Alon, R. A. Duke, H. Lefmann, V. Rödl, and R. Yuster, *The algorithmic aspects of the regularity lemma*, J. Algorithms, 16 (1994), pp. 80–109.

[3] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy, *Efficient testing of large graphs (extended abstract)*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science, New York, NY, 1999, IEEE Computer Society Press, Los Alamitos, CA, 1999, pp. 656–666.

[4] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy, *Efficient testing of large graphs*, Combinatorica, 20 (2000), pp. 451–476.

[5] N. Alon and J. Spencer, *The Probabilistic Method*, Wiley-Intersci. Ser. Discrete Math. Optim., John Wiley and Sons, New York, 1992.

[6] F. R. K. Chung and R. L. Graham, *Quasi-random set systems*, J. Amer. Math. Soc., 4 (1991), pp. 151–196.

[7] F. R. K. Chung and R. L. Graham, *Quasi-random tournaments*, J. Graph Theory, 15 (1991), pp. 173–198.

[8] F. R. K. Chung and R. L. Graham, *Quasi-random subsets of Z_n*, J. Combin. Theory Ser. A, 61 (1992), pp. 64–86.

[9] F. R. K. Chung and R. L. Graham, *Sparse quasi-random graphs*, Combinatorica, 22 (2002), pp. 217–244.

[10] F. R. K. Chung, R. L. Graham, and R. M. Wilson, *Quasi-random graphs*, Combinatorica, 9 (1989), pp. 345–362.

[11] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[12] A. Czumaj and C. Sohler, *Testing hypergraph coloring*, in Proceedings of the International Colloquium on Automata, Languages and Programming, Heraklion, Crete, Greece, 2001, pp. 493–505.

[13] A. Czygrinow, S. Poljak, and V. Rödl, *Constructive quasi-Ramsey numbers and tournament ranking*, SIAM J. Discrete Math., 12 (1999), pp. 48–63.

[14] W. Fernandez de la Vega, *MAX-CUT has a randomized approximation scheme in dense graphs*, Random Structures Algorithms, 8 (1996), pp. 187–198.

[15] R. A. Duke, H. Lefmann, and V. Rödl, *A fast approximation algorithm for computing the frequencies of subgraphs in a given graph*, SIAM J. Comput., 24 (1995), pp. 598–620.

[16] R. A. Duke and V. Rödl, *On graphs with small subgraphs of large chromatic number*, Graphs Combin., 1 (1985), pp. 91–96.

[17] P. Erdős and J. Spencer, *Probabilistic Methods in Combinatorics*, Akademiai Kiado, Budapest, 1974.

[18] A. Frieze and R. Kannan, *The regularity lemma and approximation schemes for dense problems*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Burlington, VT, 1996, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 12–20.

[19] A. Frieze and R. Kannan, *Quick approximation to matrices and applications*, Combinatorica, 19 (1999), pp. 175–220.

[20] A. Frieze and R. Kannan, *A simple algorithm for constructing Szemerédi's regularity partition*, Electronic J. Combin., 6 (1999), Research Paper 17.

[21] O. Goldreich, *Combinatorial property testing (a survey)*, in Randomization Methods in Algorithm Design (Princeton, NJ, 1997), AMS, Providence, RI, 1999, pp. 45–59.

[22] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Burlington, VT, 1996, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 339–348.

[23] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.

[24] Y. Kohayakawa and V. Rödl, *Szemerédi's regularity lemma and quasi-randomness*, in Recent Advances in Algorithms and Combinatorics, CMS Books Math./Ouvrages Math. SMC 11,

Springer, New York, 2003, pp. 289–351.
[25] J. Komlós and M. Simonovits, *Szemerédi's regularity lemma and its applications in graph theory*, in Combinatorics—Paul Erdős is Eighty, Vol. 2 (Keszthely, 1993), Bolyai Soc. Math. Stud. 2, D. Miklós, V. T. Sós, and T. Szőnyi, eds., János Bolyai Mathematical Society, Budapest, Hungary, 1996, pp. 295–352.
[26] A. Lubotzky, R. Phillips, and P. Sarnak, *Explicit expanders and the Ramanujan conjectures*, in Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, Berkeley, CA, 1986, ACM, New York, 1986, pp. 240–246.
[27] A. Lubotzky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica, 8 (1988), pp. 261–277.
[28] M. Luby and A. Wigderson, *Pairwise Independence and Derandomization*, Tech. Report 95-035, International Computer Science Institute, Berkeley, CA, 1995.
[29] J. Spencer, *Optimal ranking of tournaments*, Networks, 1 (1971), pp. 135–138.
[30] J. Spencer, *Nonconstructive methods in discrete mathematics*, in Studies in Combinatorics, G. C. Rota, ed., Mathematical Association of America, Washington, D.C., 1978, pp. 142–178.
[31] E. Szemerédi, *Regular partitions of graphs*, in Problèmes Combinatoires et Théorie des Graphes (Orsay, 1976), Colloques Internationaux CNRS 260, CNRS, Paris, 1978, pp. 399–401.
[32] A. G. Thomason, *Pseudorandom graphs*, in Random Graphs '85 (Poznań, 1985), North-Holland Math. Stud. 144, North-Holland, Amsterdam, New York, 1987, pp. 307–331.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1236–1259

DIMENSION IN COMPLEXITY CLASSES*

JACK H. LUTZ†

Abstract. A theory of resource-bounded dimension is developed using *gales*, which are natural generalizations of martingales. When the resource bound Δ (a parameter of the theory) is unrestricted, the resulting dimension is precisely the classical Hausdorff dimension (sometimes called "fractal dimension"). Other choices of the parameter Δ yield internal dimension theories in E, E_2, ESPACE, and other complexity classes, and in the class of all decidable problems. In general, if $\mathcal{C}$ is such a class, then *every* set X of languages has a dimension in $\mathcal{C}$, which is a real number $\dim(X \mid \mathcal{C}) \in [0,1]$. Along with the elements of this theory, two preliminary applications are presented:

1. For every real number $0 \leq \alpha \leq \frac{1}{2}$, the set FREQ($\leq \alpha$), consisting of all languages that asymptotically contain at most α of all strings, has dimension $\mathcal{H}(\alpha)$—the binary entropy of α—in E and in E_2.
2. For every real number $0 \leq \alpha \leq 1$, the set SIZE($\alpha \frac{2^n}{n}$), consisting of all languages decidable by Boolean circuits of at most $\alpha \frac{2^n}{n}$ gates, has dimension α in ESPACE.

Key words. circuit-size complexity, complexity classes, computational complexity, gales, Hausdorff dimension, martingales, resource-bounded dimension

AMS subject classifications. 68Q15, 03D15, 28A78

DOI. 10.1137/S0097539701417723

1. Introduction.

Since the development of resource-bounded measure in 1991 [9], the investigation of the internal, measure-theoretic structure of complexity classes has produced a rapidly growing body of new insights and results. As indicated by the survey papers [1, 3, 10, 12], this line of inquiry has shed light on a wide variety of topics in computational complexity. The ongoing fruitfulness of this research is not surprising because resource-bounded measure is a complexity-theoretic generalization of classical Lebesgue measure, which was one of the most powerful quantitative tools of twentieth-century mathematics.

In spite of this power, there are certain inherent limitations to the amount of quantitative information that resource-bounded measure can provide in computational complexity. One of these limitations arises from the resource-bounded Kolmogorov zero-one law, which was proven by Lutz [11] and has recently been strengthened by Dai [4]. For any class $\mathcal{C}$ in which resource-bounded measure is defined, and for any set X of languages that is—like most sets of interest in computational complexity—closed under finite variations, the zero-one law says that the measure of X in $\mathcal{C}$ must be 0 or 1 or undefined. A second limitation arises from the simple fact that even a measure 0 subset of a complexity class may have internal structure that we would like to elucidate quantitatively. Of course both these limitations were already present in classical Lebesgue measure theory.

In 1919, Hausdorff [7] augmented classical Lebesgue measure theory with a theory of dimension. This theory assigns to *every* subset X of a given metric space a real number $\dim_H(X)$, which is now called the *Hausdorff dimension* of X. In this paper, we are interested in the case where the metric space is the Cantor space $\mathbf{C}$, consisting

*Received by the editors June 14, 2001; accepted for publication (in revised form) November 5, 2002; published electronically August 6, 2003. This research was supported in part by National Science Foundation grants 9610461 and 9988483.
http://www.siam.org/journals/sicomp/32-5/41772.html
†Department of Computer Science, Iowa State University, Ames, IA (lutz@cs.iastate.edu).

of all decision problems (i.e., all languages $A \subseteq \{0,1\}^*$). In this case the Hausdorff dimension of a set $X \subseteq \mathbf{C}$ (which is defined precisely in section 3) is a real number $\dim_H(X) \in [0,1]$. The Hausdorff dimension is monotone, with $\dim_H(\emptyset) = 0$ and $\dim_H(\mathbf{C}) = 1$. Moreover, if $\dim_H(X) < \dim_H(\mathbf{C})$, then X is a measure 0 subset of $\mathbf{C}$. Hausdorff dimension thus overcomes both of the limitations mentioned in the preceding paragraph.

In this paper we develop *resource-bounded dimension*, which is a complexity-theoretic generalization of classical Hausdorff dimension. We carry out this generalization in two steps. We first prove a new characterization of classical Hausdorff dimension in terms of *gales*, which are natural generalizations of the martingales that are the basis of resource-bounded measure. (Our characterization can be regarded as an analogue of Ville's martingale characterization of the Lebesgue measure 0 sets [19].) We then generalize classical dimension by introducing a resource bound Δ (a parameter of the theory) and requiring the gales to be Δ-computable. We show that this induces a well-behaved notion of dimension in the corresponding class $R(\Delta)$, which is defined exactly as in resource-bounded measure. We write $\dim(X \mid R(\Delta))$ for the dimension of the set X in the class $R(\Delta)$. If Δ is unrestricted, then $R(\Delta) = \mathbf{C}$ and $\dim(X \mid R(\Delta)) = \dim(X \mid \mathbf{C})$ is precisely $\dim_H(X)$. However, other choices of Δ allow $R(\Delta)$ to be interesting complexity classes, in which case $\dim(X \mid R(\Delta))$ is a quantitative measure of the dimension of $X \cap R(\Delta)$ *as a subset of* $R(\Delta)$.

After presenting the elements of resource-bounded dimension, we present two preliminary applications of the theory. First, for each real number $\alpha \in [0,1]$, let $\mathrm{FREQ}(\leq \alpha)$ be the set of all languages that asymptotically contain at most α of all strings. (This set is defined precisely in section 5.) We prove that, for every real number $\alpha \in [0, \frac{1}{2}]$,

$$\dim(\mathrm{FREQ}(\leq \alpha) \mid \mathrm{E}) = \mathcal{H}(\alpha)$$

and

$$\dim(\mathrm{FREQ}(\leq \alpha) \mid \mathrm{E}_2) = \mathcal{H}(\alpha) \ ,$$

where $\mathrm{E} = \mathrm{DTIME}(2^{\mathrm{linear}})$, $\mathrm{E}_2 = \mathrm{DTIME}(2^{\mathrm{poly}})$, and $\mathcal{H}$ is the binary entropy function of Shannon information theory.

Our second application concerns Boolean circuit-size complexity in the complexity class $\mathrm{ESPACE} = \mathrm{DSPACE}(2^{\mathrm{linear}})$. For each real number $\alpha \in [0,1]$, let $\mathrm{SIZE}(\alpha \frac{2^n}{n})$ be the set of all languages that can be decided by Boolean circuits consisting of at most $\alpha \frac{2^n}{n}$ gates. We prove that for all $\alpha \in [0,1]$,

$$\dim\left(\mathrm{SIZE}\left(\alpha \frac{2^n}{n}\right) \middle| \mathrm{ESPACE}\right) = \alpha \ .$$

These applications are interesting because they show that resource-bounded dimension interacts informatively with information theory and Boolean circuit-size complexity. However, they are clearly only the beginning. Classical Hausdorff dimension is a sophisticated mathematical theory that has emerged as one of the most important tools for the investigation of fractal sets. (See, for example, [6] for a good introduction and overview.) Many sets of interest in computational complexity seem to have "fractal-like" structures. Resource-bounded dimension will be a useful tool for the study of such sets.

2. Preliminaries. A *decision problem* (a.k.a. *language*) is a set $A \subseteq \{0,1\}^*$. We identify each language with its characteristic sequence $[\![s_0 \in A]\!][\![s_1 \in A]\!][\![s_2 \in A]\!]\cdots$, where $s_0, s_1, s_2, \ldots$ is the standard enumeration of $\{0,1\}^*$ and $[\![\phi]\!] = \texttt{if } \phi \texttt{ then } 1 \texttt{ else } 0$. We write $A[i..j]$ for the string consisting of the ith through jth bits of (the characteristic sequence of) A. The Cantor space $\mathbf{C}$ is the set of all decision problems.

If $w \in \{0,1\}^*$ and $x \in \{0,1\}^* \cup \mathbf{C}$, then $w \sqsubseteq x$ means that w is a prefix of x, and $w \underset{\neq}{\sqsubset} x$ means that w is a proper prefix of x. The *cylinder* generated by a string $w \in \{0,1\}^*$ is $\mathbf{C}_w = \{A \in \mathbf{C} \mid w \sqsubseteq A\}$.

A *prefix set* is a language A such that no element of A is a prefix of any other element of A.

If A is a language and $n \in \mathbb{N}$, then we write $A_{=n} = A \cap \{0,1\}^n$ and $A_{\leq n} = A \cap \{0,1\}^{\leq n}$.

All logarithms in this paper are base 2.

For each $i \in \mathbb{N}$ we define a class G_i of functions from $\mathbb{N}$ into $\mathbb{N}$ as follows:

$$G_0 = \{f \mid (\exists k)(\forall^\infty n) f(n) \leq kn\},$$
$$G_{i+1} = 2^{G_i(\log n)} = \{f \mid (\exists g \in G_i)(\forall^\infty n) f(n) \leq 2^{g(\log n)}\}.$$

We also define the functions $\hat{g}_i \in G_i$ by $\hat{g}_0(n) = 2n$, $\hat{g}_{i+1}(n) = 2^{\hat{g}_i(\log n)}$. We regard the functions in these classes as growth rates. In particular, G_0 contains the linearly bounded growth rates and G_1 contains the polynomially bounded growth rates. It is easy to show that each G_i is closed under composition, that each $f \in G_i$ is $o(\hat{g}_{i+1})$, and that each $\hat{g}_i$ is $o(2^n)$. Thus G_i contains superpolynomial growth rates for all $i > 1$, but all growth rates in the G_i-hierarchy are subexponential.

Within the class DEC of all decidable languages, we are interested in the exponential complexity classes $\mathrm{E}_i = \mathrm{DTIME}(2^{G_{i-1}})$ and $\mathrm{E}_i\mathrm{SPACE} = \mathrm{DSPACE}(2^{G_{i-1}})$ for $i \geq 1$. The much-studied classes $\mathrm{E} = \mathrm{E}_1 = \mathrm{DTIME}(2^{\mathrm{linear}})$, $\mathrm{E}_2 = \mathrm{DTIME}(2^{\mathrm{polynomial}})$, and $\mathrm{ESPACE} = \mathrm{E}_1\mathrm{SPACE} = \mathrm{DSPACE}(2^{\mathrm{linear}})$ are of particular interest.

We use the following classes of functions:

$$\mathrm{all} = \{f \mid f : \{0,1\}^* \to \{0,1\}^*\},$$
$$\mathrm{comp} = \{f \in \mathrm{all} \mid f \text{ is computable}\},$$
$$\mathrm{p}_i = \{f \in \mathrm{all} \mid f \text{ is computable in } G_i \text{ time}\} \ (i \geq 1),$$
$$\mathrm{p}_i\mathrm{space} = \{f \in \mathrm{all} \mid f \text{ is computable in } G_i \text{ space}\} \ (i \geq 1).$$

(The length of the output *is* included as part of the space used in computing f.) We write p for p_1 and pspace for $\mathrm{p}_1$space. Throughout this paper, Δ and Δ' denote one of the classes all, comp, p_i $(i \geq 1)$, or $\mathrm{p}_i$space $(i \geq 1)$.

A *constructor* is a function $\delta : \{0,1\}^* \to \{0,1\}^*$ that satisfies $x \underset{\neq}{\sqsubset} \delta(x)$ for all x. The *result* of a constructor δ (i.e., the language *constructed* by δ) is the unique language $R(\delta)$ such that $\delta^n(\lambda) \sqsubseteq R(\delta)$ for all $n \in \mathbb{N}$. Intuitively, δ constructs $R(\delta)$ by starting with λ and then iteratively generating successively longer prefixes of $R(\delta)$. We write $R(\Delta)$ for the set of languages $R(\delta)$ such that δ is a constructor in Δ. The following facts are the reason for our interest in the above-defined classes of functions:

$R(\mathrm{all}) = \mathbf{C}$.
$R(\mathrm{comp}) = \mathrm{DEC}$.
For $i \geq 1$, $R(\mathrm{p}_i) = \mathrm{E}_i$.
For $i \geq 1$, $R(\mathrm{p}_i\mathrm{space}) = \mathrm{E}_i\mathrm{SPACE}$.

If D is a discrete domain, then a function $f : D \longrightarrow [0,\infty)$ is Δ-*computable* if there is a function $\hat{f} : \mathbb{N} \times D \longrightarrow \mathbb{Q} \cap [0,\infty)$ such that $|\hat{f}(r,x) - f(x)| \leq 2^{-r}$ for all

$r \in \mathbb{N}$ and $x \in D$ and $\hat{f} \in \Delta$ (with r coded in unary and the output coded in binary). We say that f is *exactly* Δ*-computable* if $f : D \longrightarrow \mathbb{Q} \cap [0, \infty)$ and $f \in \Delta$.

3. Gales and Hausdorff dimension. In this section we introduce gales and supergales, which are generalizations of martingales and supermartingales, and use these to give a new characterization of classical Hausdorff dimension.

DEFINITION 3.1. *Let* $s \in [0, \infty)$.

1. *An* s-supergale *is a function* $d : \{0,1\}^* \longrightarrow [0, \infty)$ *that satisfies the condition*

$$d(w) \geq 2^{-s}[d(w0) + d(w1)] \tag{3.1}$$

   for all $w \in \{0,1\}^*$.
2. *An* s-gale *is an* s*-supergale that satisfies* (3.1) *with equality for all* $w \in \{0,1\}^*$.
3. *A* supermartingale *is a* 1*-supergale.*
4. *A* martingale *is a* 1*-gale.*

LEMMA 3.2. *Let* $s \in [0, \infty)$. *If* d *is an* s*-supergale and* $B \subseteq \{0,1\}^*$ *is a prefix set, then for all* $w \in \{0,1\}^*$,

$$\sum_{u \in B} 2^{-s|u|} d(wu) \leq d(w).$$

Proof. We first use induction on n to prove that for all $n \in \mathbb{N}$, the lemma holds for all prefix sets $B \subseteq \{0,1\}^{\leq n}$. For $n = 0$, this is trivial. Assume that it holds for n, and let $A \subseteq \{0,1\}^{\leq n+1}$ be a prefix set. Let

$$A' = \{u \in \{0,1\}^n \mid u0 \in A \text{ or } u1 \in A\},$$

and let

$$B = A_{\leq n} \cup A'.$$

Note that B is a prefix set and $A_{\leq n} \cap A' = \emptyset$ (because A is a prefix set). Also, for all $w \in \{0,1\}^*$,

$$\begin{aligned}
\sum_{u \in A_{=n+1}} 2^{-s|u|} d(wu) &= 2^{-s(n+1)} \sum_{u \in A_{=n+1}} d(wu) \\
&\leq 2^{-s(n+1)} \sum_{u \in A'} [d(wu0) + d(wu1)] \\
&\leq 2^{-s(n+1)} \sum_{u \in A'} 2^s d(wu) \\
&= \sum_{u \in A'} 2^{-s|u|} d(wu).
\end{aligned}$$

Since $B \subseteq \{0,1\}^{\leq n}$, it follows by the induction hypothesis that for all $w \in \{0,1\}^*$,

$$\begin{aligned}
\sum_{u \in A} 2^{-s|u|} d(wu) &= \sum_{u \in A_{\leq n}} 2^{-s|u|} d(wu) + \sum_{u \in A_{=n+1}} 2^{-s|u|} d(wu) \\
&\leq \sum_{u \in A_{\leq n}} 2^{-s|u|} d(wu) + \sum_{u \in A'} 2^{-s|u|} d(wu) \\
&= \sum_{u \in B} 2^{-s|u|} d(wu) \\
&\leq d(w).
\end{aligned}$$

This completes the proof that for all $n \in \mathbb{N}$, the lemma holds for all prefix sets $B \subseteq \{0,1\}^{\leq n}$.

To complete the proof of the lemma, let B be an arbitrary prefix set. Then for all $w \in \{0,1\}^*$,

$$\sum_{u \in B} 2^{-s|u|} d(wu) = \sup_{n \in \mathbb{N}} \sum_{u \in B_{\leq n}} 2^{-s|u|} d(wu) \leq d(w). \qquad \square$$

COROLLARY 3.3. *Let $s \in [0,\infty)$, $0 < \alpha \in \mathbb{R}$, and $w \in \{0,1\}^*$. If d is an s-supergale such that $d(w) > 0$ and $B \subseteq \{0,1\}^*$ is a prefix set such that $d(wu) \geq \alpha 2^{(s-1)|u|} d(w)$ for all $u \in B$, then*

$$\sum_{u \in B} 2^{-|u|} \leq \frac{1}{\alpha}.$$

Proof. Assume the hypothesis. Then by Lemma 3.2,

$$d(w) \geq \sum_{u \in B} 2^{-s|u|} d(wu) \geq \alpha d(w) \sum_{u \in B} 2^{-|u|},$$

whence the corollary follows. $\square$

COROLLARY 3.4. *Let d be an s-supergale, where $s \in [0,1]$. Then for all $w \in \{0,1\}^*$, $l \in \mathbb{N}$, and $0 < \alpha \in \mathbb{R}$, there are fewer than $\frac{2^l}{\alpha}$ strings $u \in \{0,1\}^l$ for which*

$$\max_{v \sqsubseteq u} 2^{(1-s)|v|} d(wv) > \alpha d(w).$$

In particular, there is at least one string $u \in \{0,1\}^l$ such that $d(wv) \leq 2^{(s-1)|v|} d(w)$ for all $v \sqsubseteq u$.

Proof. Let d, s, w, l, and α be as given, and let

$$A = \left\{ u \in \{0,1\}^l \mid \max_{v \sqsubseteq u} 2^{(1-s)|v|} d(wv) > \alpha d(w) \right\}.$$

If $A = \emptyset$, the corollary is trivially affirmed, so assume that $A \neq \emptyset$. (Note that this implies $d(w) > 0$.) Let B be the set of all $v \in \{0,1\}^{\leq l}$ such that $2^{(1-s)|v|} d(wv) > \alpha d(w)$ but $2^{(1-s)|v'|} d(wv') \leq \alpha d(w)$ for all $v' \underset{\neq}{\sqsubset} v$. Then B is a prefix set, and

$$A = \{u \in \{0,1\}^l \mid (\exists v \sqsubseteq u) v \in B\},$$

so

$$|A| = \sum_{v \in B} 2^{l-|v|} = 2^l \sum_{v \in B} 2^{-|v|}.$$

Let

$$\alpha' = \min_{v \in B} 2^{(1-s)|v|} \frac{d(wv)}{d(w)},$$

and note that $\alpha < \alpha' < \infty$. Then B is a prefix set such that $d(wv) \geq \alpha' 2^{(s-1)|v|} d(w)$ for all $v \in B$, so Corollary 3.3 tells us that

$$|A| = 2^l \sum_{v \in B} 2^{-|v|} \leq \frac{2^l}{\alpha'} < \frac{2^l}{\alpha}.$$

This proves the main assertion of the corollary. The last sentence of the corollary follows immediately by taking $\alpha = 1$. □

COROLLARY 3.5. *If d is an s-supergale, where $s \in [0, \infty)$, then for all $w, u \in \{0,1\}^*$,*

$$d(wu) \leq 2^{s|u|} d(w).$$

Proof. Let d, s, w, and u be as given, and let $l = |u|$. Let $\beta > 2^{s|u|}$ be arbitrary. It suffices to show that $d(wu) \leq \beta d(w)$.

Let $\alpha = \beta 2^{(1-s)l}$. Then, for all $v \in \{0,1\}^l$,

$$d(wv) > \beta d(w) \Leftrightarrow d(wv) > \alpha 2^{(s-1)l} d(w),$$

so Corollary 3.4 tells us that there are fewer than $\frac{2^l}{\alpha}$ strings $v \in \{0,1\}^l$ for which $d(wv) > \beta d(w)$. Since $\frac{2^l}{\alpha} = \frac{2^{s|u|}}{\beta} < 1$, it follows that $d(wu) \leq \beta d(w)$. □

Observation 3.6. Let $s \in [0, \infty)$. For each $k \in \mathbb{N}$, let d_k be an s-gale, and let $a_k \in [0, \infty)$.

1. For each $n \in \mathbb{Z}^+$, $\sum_{k=0}^{n-1} a_k d_k$ is an s-gale.
2. If $\sum_{k=0}^{\infty} a_k d_k(\lambda) < \infty$, then $\sum_{k=0}^{\infty} a_k d_k$ is an s-gale.

DEFINITION 3.7. *Let d be an s-supergale, where $s \in [0, \infty)$.*

1. *We say that d* succeeds *on a language $A \in \mathbf{C}$ if*

$$\limsup_{n \to \infty} d(A[0 \ldots n-1]) = \infty.$$

2. *The* success set *of d is*

$$S^\infty[d] = \{A \in \mathbf{C} \mid d \text{ succeeds on } A\}.$$

We now review the classical definition of Hausdorff dimension. Since we are primarily interested in the computational complexities of decision problems, we focus on Hausdorff dimension in the Cantor space $\mathbf{C}$.

For each $k \in \mathbb{N}$, we let $\mathcal{A}_k$ be the collection of all prefix sets A such that $A_{<k} = \emptyset$. For each $X \subseteq \mathbf{C}$, we then let

$$\mathcal{A}_k(X) = \left\{ A \in \mathcal{A}_k \;\middle|\; X \subseteq \bigcup_{w \in A} \mathbf{C}_w \right\}.$$

If $A \in \mathcal{A}_k(X)$, then we say that the prefix set A *covers* the set X. For $X \in \mathbf{C}$, $s \in [0, \infty)$, and $k \in \mathbb{N}$, we then define

$$H_k^s(X) = \inf_{A \in \mathcal{A}_k(X)} \sum_{w \in A} 2^{-s|w|}.$$

Digression. Readers familiar with Hausdorff dimension may prefer to regard it as arising from a measure or, more commonly, a metric on the underlying space. If we regard Hausdorff dimension as arising from a measure on $\mathbf{C}$, then the term $2^{-s|w|}$ in the above sum is interpreted as $\mu(\mathbf{C}_w)^s$, where $\mu(\mathbf{C}_w) = 2^{-|w|}$ is the Lebesgue measure of the cylinder $\mathbf{C}_w$. If we instead regard Hausdorff dimension as arising from a metric on $\mathbf{C}$, then the metric is

$$d(A, B) = 2^{-\min\{n \in \mathbb{N} \mid A[n] \neq B[n]\}}$$

(where $\min \emptyset = \infty$ so $d(A, A) = 0$), and the term $2^{-s|w|}$ is interpreted as $\mathrm{diam}(\mathbf{C}_w)^s$, where

$$\mathrm{diam}(X) = \sup_{A,B \in X} d(A, B)$$

is the *diameter* of a set $X \subseteq \mathbf{C}$. Such interpretations may provide context, but our technical development does not use them.

DEFINITION 3.8. *For $s \in [0, \infty)$ and $X \subseteq \mathbf{C}$, the* s-dimensional Hausdorff cylinder outer measure *of X is*

$$H^s(X) = \lim_{k \to \infty} H^s_k(X).$$

Since $H^s_k(X)$ is nondecreasing in k, this limit $H^s(X)$ exists, though it may be infinite. In fact, it is well known that for every set $X \subseteq \mathbf{C}$, there is a real number $s^* \in [0, 1]$ with the following two properties:

(i) For all $0 \leq s < s^*$, $H^s(X) = \infty$.

(ii) For all $s > s^*$, $H^s(X) = 0$.

As the following definition states, this number s^* is the Hausdorff dimension of X.

DEFINITION 3.9. *The* Hausdorff dimension *of a set $X \subseteq \mathbf{C}$ is*

$$\dim_H(X) = \inf\{s \in [0, \infty) \mid H^s(X) = 0\}.$$

Notation. For $X \subseteq \mathbf{C}$, let $\mathcal{G}(X)$ be the set of all $s \in [0, \infty)$ such that there is an s-gale d for which $X \subseteq S^\infty[d]$.

The following theorem gives a new characterization of classical Hausdorff dimension.

THEOREM 3.10 (gale characterization of Hausdorff dimension). *For all $X \subseteq \mathbf{C}$, $\dim_H(X) = \inf \mathcal{G}(X)$.*

Proof. It suffices to show that for all $s \in [0, \infty)$,

$$H^s(X) = 0 \Leftrightarrow s \in \mathcal{G}(X).$$

First, assume that $H^s(X) = 0$. Then $H^s_0(X) = 0$, which implies that for each $r \in \mathbb{N}$, there is a prefix set $A_r \in \mathcal{A}_0(X)$ such that $\sum_{w \in A_r} 2^{-s|w|} \leq 2^{-r}$. For each $r \in \mathbb{N}$, then, fix such a prefix set A_r, and define a function $d_r : \{0,1\}^* \longrightarrow [0, \infty)$ as follows. Let $w \in \{0,1\}^*$. If there exists $v \sqsubseteq w$ such that $v \in A_r$, then $d_r(w) = 2^{(s-1)(|w|-|v|)}$. Otherwise,

$$d_r(w) = \sum_{\substack{u \\ wu \in A_r}} 2^{-s|u|}.$$

It is routine to verify that the following conditions hold for all $r \in \mathbb{N}$:

(i) d_r is an s-gale.

(ii) $d_r(\lambda) \leq 2^{-r}$.

(iii) For all $w \in A_r$, $d_r(w) = 1$.

Let $d = \sum_{r=0}^\infty 2^r d_{2r}$. By Observation 3.6, d is an s-gale. To see that $X \subseteq S^\infty[d]$, let $B \in X$, and let $r \in \mathbb{N}$ be arbitrary. Since the prefix set A_{2r} covers X, there exists $w \in A_{2r}$ such that $w \sqsubseteq B$. Then by (iii) above, $d(w) \geq 2^r d_{2r}(w) = 2^r$. Since $r \in \mathbb{N}$ is arbitrary, this shows that $B \in S^\infty$, confirming that $X \subseteq S^\infty[d]$.

We have now shown that d is an s-gale such that $X \subseteq S^\infty[d]$, whence $s \in \mathcal{G}(X)$.

Conversely, assume that $s \in \mathcal{G}(X)$. To see that $H^s(X) = 0$, let $k, r, \in \mathbb{N}$. It suffices to show that $H^s(X) \leq 2^{-r}$. If $X = \emptyset$, this is trivial, so assume that $X \neq \emptyset$.

Since $s \in \mathcal{G}(X)$, there is an s-gale d such that $X \subseteq S^\infty[d]$. Note that $d(\lambda) > 0$ because $X \neq 0$. Let

$$a = 1 + \max\{d(w) \mid w \in \{0,1\}^{\leq k}\},$$

and let

$$A = \left\{ w \in \{0,1\}^* \middle| d(w) \geq 2^r a \text{ and } (\forall v) \left[v \underset{\neq}{\sqsubset} w \Rightarrow d(v) < 2^r a \right] \right\}.$$

It is clear that A is a prefix set with $A_{<k} = \emptyset$, so $A \in \mathcal{A}_k$. It is also clear that

$$X \subseteq S^\infty[d] \subseteq \bigcup_{w \in A} \mathbf{C}_w,$$

whence $A \in \mathcal{A}_k(X)$. By Lemma 3.2 and the definition of A, we have

$$d(\lambda) \geq \sum_{w \in A} 2^{-s|w|} d(w) \geq 2^r d(\lambda) \sum_{w \in A} 2^{-s|w|}.$$

Since $A \in \mathcal{A}_k(X)$ and $d(\lambda) > 0$, it follows that

$$H^s_k(X) \leq \sum_{w \in A} 2^{-s|w|} \leq 2^{-r}. \qquad \square$$

It is clear from the proof of Theorem 3.10 that the s-dimensional Hausdorff cylinder outer measure $H^s(X)$ can also be characterized in terms of s-gales, but we refrain from elaborating here.

4. Dimension in complexity classes. Motivated by the gale characterization of classical Hausdorff dimension, we now use resource-bounded gales to develop dimension in complexity classes. As with resource-bounded measure [9], our development contains a parameter Δ—the resource bound—which may be any one of the classes all, comp, p, pspace, p_2, $\mathrm{p}_2$space, etc., defined in section 2.

Notation. For $X \subseteq \mathbf{C}$, let $\mathcal{G}_\Delta(X)$ be the set of all $s \in [0, \infty)$ such that there is a Δ-computable s-gale d for which $X \subseteq S^\infty[d]$.

DEFINITION 4.1. *Let* $X \subseteq \mathbf{C}$.

1. *The* Δ-dimension *of* X *is* $\dim_\Delta(X) = \inf \mathcal{G}_\Delta(X)$.
2. *The* dimension of X in $R(\Delta)$ *is* $\dim(X \mid R(\Delta)) = \dim_\Delta(X \cap R(\Delta))$.

Note that $\dim_\Delta(X)$ and $\dim(X \mid R(\Delta))$ are defined for *every* set $X \subseteq \mathbf{C}$. We call $\dim_{\mathrm{comp}}(X)$ and $\dim_{\mathrm{p}}(X)$ the *computable dimension* of X and the *feasible dimension* of X, respectively.

The following observations are elementary but useful.

Observation 4.2.

1. For all $X \subseteq Y \subseteq \mathbf{C}$,

$$\dim_\Delta(X) \leq \dim_\Delta(Y)$$

and

$$\dim(X \mid R(\Delta)) \leq \dim(Y \mid R(\Delta)).$$

2. If Δ and Δ' are resource bounds such that $\Delta \subseteq \Delta'$, then for all $X \subseteq \mathbf{C}$,
$$\dim_{\Delta'}(X) \leq \dim_{\Delta}(X).$$
3. For all $X \subseteq \mathbf{C}$, $0 \leq \dim(X \mid R(\Delta)) \leq \dim_{\Delta}(X) \leq 1$.
4. For all $X \subseteq \mathbf{C}$, $\dim(X \mid \mathbf{C}) = \dim_{\mathrm{all}}(X) = \dim_H(X)$.
5. For all $X \subseteq \mathbf{C}$,
$$\dim_{\Delta}(X) < 1 \Rightarrow \mu_{\Delta}(X) = 0$$
and
$$\dim(X \mid R(\Delta)) < 1 \Rightarrow \mu(X \mid R(\Delta)) = 0.$$

Proof. Observations 1, 2, 4, and 5 follow immediately from the definitions. For observation 3, it suffices to show that $\dim_{\Delta}(\mathbf{C}) \leq 1$. For this, fix $s > 1$ such that 2^s is rational. Then the function
$$d : \{0,1\}^* \to [0, \infty)$$
$$d(w) = 2^{(s-1)|w|}$$
is a Δ-computable s-gale such that $S^{\infty}[d] = \mathbf{C}$, so $s \in \mathcal{G}_{\Delta}(\mathbf{C})$, and thus $\dim_{\Delta}(\mathbf{C}) \leq s$. Since this holds for all $s > 1$ with 2^s rational, it follows that $\dim_{\Delta}(\mathbf{C}) \leq 1$. □

The fifth observation above shows that resource-bounded dimension offers a quantitative classification of sets that have resource-bounded measure 0. However, it should be emphasized that this classification is in terms of dimension, which is very different from measure. For example, Lemma 4.9 and its corollaries exhibit properties of dimension that contrast sharply with those of measure.

To proceed further, we need a little bit of technical machinery concerning the computability of gales and supergales.

Observation 4.3. If d is a Δ-computable s-gale, then the real number 2^s is Δ-computable.

LEMMA 4.4. *If d is a Δ-computable s-supergale and 2^s is Δ-computable, then there is a Δ-computable s-gale $\tilde{d}$ such that $S^{\infty}[d] \subseteq S^{\infty}[\tilde{d}]$. Moreover, if d is exactly Δ-computable and 2^s is rational, then $\tilde{d}$ is exactly Δ-computable.*

Proof. Assume the hypothesis. Define
$$\tilde{d} : \{0,1\}^* \longrightarrow [0, \infty)$$
$$\tilde{d}(\lambda) = d(\lambda),$$
$$\tilde{d}(w0) = \frac{1}{2}[2^s \tilde{d}(w) + d(w0) - d(w1)],$$
$$\tilde{d}(w1) = \frac{1}{2}[2^s \tilde{d}(w) - d(w0) + d(w1)].$$

Then $\tilde{d}$ is clearly a Δ-computable s-gale, and an easy induction shows that $\tilde{d}(w) \geq d(w)$ for all $w \in \{0,1\}^*$, whence $S^{\infty}[d] \subseteq S^{\infty}[\tilde{d}]$. Moreover, if d is exactly Δ-computable and 2^s is rational, then it is clear that $\tilde{d}$ is exactly Δ-computable. □

Observation 4.5. If d is an s-supergale and $s' \geq s$, then d is an s'-supergale.

COROLLARY 4.6. *Let $X \subseteq \mathbf{C}$ and $s' \geq s \geq 0$. If $s \in \mathcal{G}_\Delta(X)$ and $2^{s'}$ is Δ-computable, then $s' \in \mathcal{G}_\Delta(X)$.*

Proof. This follows immediately from Lemma 4.4 and Observation 4.5. □

Note that Corollary 4.6 implies that $\mathcal{G}_\Delta(X)$ is a dense subset of the interval $[\dim_\Delta(X), \infty)$.

The following consequence of Lemma 4.4 says that supergales can be used to establish upper bounds on dimension.

COROLLARY 4.7. *Let $X \subseteq \mathbf{C}$ and $s \in [0, \infty)$. If there is a Δ-computable s-supergale d such that $X \subseteq S^\infty[d]$, then $\dim_\Delta(X) \leq s$.*

Proof. Assume the hypothesis, and let $s' \geq s$ be arbitrary such that 2^s is Δ-computable. By Observation 4.5, d is a Δ-computable s'-supergale with $X \subseteq S^\infty[d]$. It follows by Lemma 4.4 that $s' \in \mathcal{G}_\Delta(X)$, whence $\dim_\Delta(X) \leq s'$. Since this holds for all $s' \geq s$ with $2^{s'}$ Δ-computable, it follows that $\dim_\Delta(X) \leq s$. □

LEMMA 4.8 (exact computation lemma). *If d is a Δ-computable s-supergale and 2^s is rational, then there is an exactly Δ-computable s-gale $\tilde{d}$ such that $S^\infty[d] \subseteq S^\infty[\tilde{d}]$.*

Proof. Assume the hypothesis. If $s = 0$, then $S^\infty[d] = \emptyset$ and the conclusion holds trivially, so assume that $s > 0$. By Lemma 4.4, it suffices to show that there is an exactly computable s-supergale $\tilde{d}$ such that $S^\infty[d] \subseteq S^\infty[\tilde{d}]$.

Since d is Δ-computable, there is an exactly Δ-computable function $\hat{d} : \{0,1\}^* \times \mathbb{N} \longrightarrow \mathbb{Q} \cap [0, \infty)$ such that for all $w \in \{0,1\}^*$ and $r \in \mathbb{N}$, $|\hat{d}(w, r) - d(w)| \leq 2^{-r}$. Let

$$a = 1 + \left\lceil \log \frac{1}{1 - 2^{-s}} \right\rceil,$$

so that $2^{1-a} \leq 1 - 2^{-s}$, and define

$$\tilde{d} : \{0,1\}^* \longrightarrow \mathbb{Q} \cap [0, \infty)$$

$$\tilde{d}(w) = \hat{d}(w, |w| + a) + 2^{-|w|}.$$

It is clear that $\tilde{d}$ is exactly Δ-computable. Also, for all $w \in \{0,1\}^*$,

$$\begin{aligned}
&\tilde{d}(w) - 2^{-s}[\tilde{d}(w0) + \tilde{d}(w1)] \\
&\geq d(w) - 2^{-(|w|+a)} + 2^{-|w|} - 2^{-s}\left[d(w0) + d(w1) + 2\left[2^{-(|w|+a+1)} + 2^{-(|w|+1)}\right]\right] \\
&= d(w) - 2^{-s}[d(w0) + d(w1)] + 2^{-|w|}[1 - 2^{-s} - (1 + 2^{-s})2^{-a}] \\
&\geq 2^{-|w|}[1 - 2^{-s} - 2^{1-a}] \\
&\geq 0\ ,
\end{aligned}$$

so $\tilde{d}$ is an s-supergale. Finally, for all $w \in \{0,1\}^*$,

$$\tilde{d}(w) - d(w) \geq 2^{-|w|} - 2^{-(|w|+a)} > 0,$$

so $S^\infty[d] \subseteq S^\infty[\tilde{d}]$. □

LEMMA 4.9. *For all $X, Y \subseteq \mathbf{C}$,*

$$\dim_\Delta(X \cup Y) = \max\{\dim_\Delta(X), \dim_\Delta(Y)\}$$

and

$$\dim(X \cup Y \mid R(\Delta)) = \max\{\dim(X \mid R(\Delta)), \dim(Y \mid R(\Delta))\}.$$

Proof. The second identity follows from the first, so by Observation 4.2 it suffices to show that

$$\dim_\Delta(X \cup Y) \leq \max\{\dim_\Delta(X), \dim_\Delta(Y)\}.$$

For this, choose an arbitrary $s > \max\{\dim_\Delta(X), \dim_\Delta(Y)\}$ such that 2^s is Δ-computable. By Corollary 4.6, $s \in \mathcal{G}_\Delta(X) \cap \mathcal{G}_\Delta(Y)$, so there exist Δ-computable s-gales d_X and d_Y such that $X \subseteq S^\infty[d_X]$ and $Y \subseteq S^\infty[d_Y]$. Let $d = d_X + d_Y$. Then d is clearly Δ-computable , and d is an s-gale by Observation 3.6. It is clear that $X \cup Y \subseteq S^\infty[d]$, whence $s \in \mathcal{G}_\Delta(X \cup Y)$. It follows that $\dim_\Delta(X \cup Y) < s$. Since s is arbitrary here, we have shown that $\dim_\Delta(X \cup Y) \leq \max\{\dim_\Delta(X), \dim_\Delta(Y)\}$. □

Lemma 4.9 has the following immediate consequence.

COROLLARY 4.10. *For all $X \in \mathbf{C}$ and $n \in \mathbb{N}$,*

$$\dim_\Delta(X) = \max_{w \in \{0,1\}^n} \dim_\Delta(X \cap \mathbf{C}_w)$$

and

$$\dim(X \mid R(\Delta)) = \max_{w \in \{0,1\}^n} \dim(X \cap \mathbf{C}_w \mid R(\Delta)).$$

A set $X \subseteq \mathbf{C}$ is *closed under finite variations* if for every $A \in X$ and every finite set $D \subseteq \{0,1\}^*$, we have $A \triangle D \in X$, where $A \triangle D = (A - D) \cup (D - A)$ is the symmetric difference of A and D. A set X with this property is called a *tail set*.

COROLLARY 4.11. *If $X \subseteq \mathbf{C}$ is a tail set, then for all $w \in \{0,1\}^*$,*

$$\dim_\Delta(X \cap \mathbf{C}_w) = \dim_\Delta(X)$$

and

$$\dim(X \cap \mathbf{C}_w \mid R(\Delta)) = \dim(X \mid R(\Delta)).$$

Lemma 4.9 can be extended to countable unions, provided that these unions are sufficiently constructive.

DEFINITION 4.12. *Let $X, X_0, X_1, X_2, \ldots \subseteq \mathbf{C}$.*

1. *X is a Δ-union of the Δ-dimensioned sets $X_0, X_1, X_2, \ldots$ if $X = \bigcup_{k=0}^\infty X_k$ and for each $s > \sup_{k \in \mathbb{N}} \dim_\Delta(X_k)$ with 2^s rational, there is a function $d : \mathbb{N} \times \{0,1\}^* \to [0, \infty)$ with the following properties:*
   (i) *d is Δ-computable.*
   (ii) *For each $k \in \mathbb{N}$, if we write $d_k(w) = d(k, w)$, then the function d_k is an s-gale.*
   (iii) *For each $k \in \mathbb{N}$, $X_k \subseteq S^\infty[d_k]$.*
2. *X is a Δ-union of the sets $X_0, X_1, X_2, \ldots$ dimensioned in $R(\Delta)$ if $X = \bigcup_{k=0}^\infty X_k$ and $X \cap R(\Delta)$ is a Δ-union of the Δ-dimensional sets $X_0 \cap R(\Delta)$, $X_1 \cap R(\Delta)$, $X_2 \cap R(\Delta), \ldots$.*

LEMMA 4.13. *Let $X, X_0, X_1, X_2, \ldots \subseteq \mathbf{C}$.*

1. *If X is a Δ-union of the Δ-dimensioned sets $X_0, X_1, X_2, \ldots$, then*

$$\dim_\Delta(X) = \sup_{k \in \mathbb{N}} \dim_\Delta(X_k).$$

2. *If X is a Δ-union of the sets $X_0, X_1, X_2, \ldots$ dimensioned in $R(\Delta)$, then*

$$\dim(X \mid R(\Delta)) = \sup_{k \in \mathbb{N}} \dim(X_k \mid R(\Delta)).$$

Proof. It suffices to prove 1, since 2 follows immediately from 1. Assume the hypothesis of 1, and let $s > \sup_{k\in\mathbb{N}} \dim_\Delta(X_K)$ be arbitrary with 2^s rational and $s < 2$. By Observation 4.2, it suffices to show that $\dim_\Delta(X) \leq s$.

Since X is a union of the Δ-dimensioned sets $X_0, X_1, X_2, \ldots$, there is a Δ-computable function $d : \mathbb{N} \times \{0,1\}^* \longrightarrow [0,\infty)$ such that each d_k is an s-gale with $X_k \subseteq S^\infty[d_k]$. Without loss of generality (modifying d if necessary), we can assume that each $d_k(\lambda) \leq 1$.

Let $\tilde{d} = \sum_{k=0}^\infty 2^{-k} d_k$. By Observation 3.6, $\tilde{d}$ is an s-gale. Since d is Δ-computable, there is a function $\hat{d} : \mathbb{N} \times \mathbb{N} \times \{0,1\}^* \longrightarrow \mathbb{Q} \cap [0,\infty)$ such that $\hat{d} \in \Delta$ and for all $r, k \in \mathbb{N}$ and $w \in \{0,1\}^*$, $|\hat{d}(r,k,w) - d(k,w)| \leq 2^{-r}$. Define

$$\hat{\tilde{d}} : \mathbb{N} \times \{0,1\}^* \longrightarrow \mathbb{Q} \cap [0,\infty)$$

by

$$\hat{\tilde{d}}(r,w) = \sum_{k=0}^{r+2|w|+1} 2^{-k} \hat{d}(r+2,k,w).$$

Then $\hat{\tilde{d}} \in \Delta$ and for all $r \in \mathbb{N}$ and $w \in \{0,1\}^*$,

$$|\hat{\tilde{d}}(r,w) - \tilde{d}(w)| \leq |\tilde{d}(w) - a| + |a - \hat{\tilde{d}}(w)|,$$

where $a = \sum_{k=0}^{r+2|w|+1} 2^{-k} d_k(w)$. By Corollary 3.5,

$$\begin{aligned} |\tilde{d}(w) - a| &= \sum_{k=r+2|w|+2}^{\infty} 2^{-k} d_k(w) \\ &\leq \sum_{k=r+2|w|+2}^{\infty} 2^{-k} 2^{s|w|} d_k(\lambda) \\ &\leq \sum_{k=r+2|w|+2}^{\infty} 2^{2|w|-k} \\ &= 2^{-(r+1)}. \end{aligned}$$

Also,

$$\begin{aligned} |a - \hat{\tilde{d}}(w)| &\leq \sum_{k=0}^{r+2|w|+1} 2^{-k} |\tilde{d}(r+2,k,w) - d(k,w)| \\ &\leq \sum_{k=0}^{\infty} 2^{-(k+r+2)} \\ &= 2^{-(r+1)}. \end{aligned}$$

It follows that for all $r \in \mathbb{N}$ and $w \in \{0,1\}^*$,

$$|\hat{\tilde{d}}(r,w) - \tilde{d}| \leq 2^{-r},$$

whence $\hat{\tilde{d}}$ testifies that $\tilde{d}$ is Δ-computable. It is clear that $X = \bigcup_{k=0}^\infty X_k \subseteq \bigcup_{k=0}^\infty S^\infty[d_k] \subseteq S^\infty[\tilde{d}]$, so it follows that $\dim_\Delta(X) \leq s$. ∎

We now note that finite sets of languages have resource-bounded dimension 0, provided that the languages themselves do not exceed the resource bound.

LEMMA 4.14. *If $X \subseteq R(\Delta)$ is finite, then* $\dim(X \mid R(\Delta)) = \dim_\Delta(X) = 0$.

Proof. By Lemma 4.9, it suffices to prove this for singleton sets $X \subseteq R(\Delta)$, so assume that $X = \{A\}$, where $A \in R(\Delta)$. Let $s > 0$ be arbitrarily small with 2^s rational. It suffices to show that $\dim_\Delta(X) \leq s$. Define

$$d : \{0,1\}^* \longrightarrow [0,\infty)$$

$$d(w) = \begin{cases} 2^{s|w|} & \text{if } w \sqsubseteq A, \\ 0 & \text{if } w \not\sqsubseteq A. \end{cases}$$

Then $d \in \Delta$ (because $A \in R(\Delta)$ and 2^s is rational) and it is clear that d is an s-gale that succeeds on A. Thus $\dim_\Delta(X) \leq s$. □

Lemma 4.14 can be extended to subsets of $R(\Delta)$ that are "countable within the resource bound Δ" in the following sense. A set $X \subseteq \mathbf{C}$ is Δ-*countable* if there is a function $\delta : \mathbb{N} \times \{0,1\}^* \to \{0,1\}^*$ with the following properties:

(i) $\delta \in \Delta$.

(ii) For each $k \in \mathbb{N}$, if we write $\delta_k(w) = \delta(k,w)$, then the function δ_k is a constructor.

(iii) $X = \{R(\delta_k) \mid k \in \mathbb{N}\}$.

It is clear that if X is Δ-countable, then $X \subseteq R(\Delta)$. It was shown in [9] that every Δ-countable set has Δ-measure 0. We now show that more is true.

LEMMA 4.15. *If $X \subseteq \mathbf{C}$ is Δ-countable, then* $\dim(X \mid R(\Delta)) = \dim_\Delta(X) = 0$.

Proof. Let the function $\delta : \mathbb{N} \times \{0,1\}^* \longrightarrow \{0,1\}^*$ testify that X is Δ-countable. By Lemmas 4.13 and 4.14, it suffices to show that X is a Δ-union of the Δ-dimension 0 singleton sets $\{R(\delta_0)\}, \{R(\delta_1)\}, \{R(\delta_2)\}, \ldots$. For this, let $s > 0$ with 2^s rational, and define

$$d : \mathbb{N} \times \{0,1\}^* \longrightarrow [0,\infty)$$

$$d(k,w) = \begin{cases} 2^{s|w|} & \text{if } w \sqsubseteq R(\delta_k), \\ 0 & \text{if } w \not\sqsubseteq R(\delta_k). \end{cases}$$

Then d clearly has the required properties. □

If $\Delta =$ all, then Lemma 4.15 is the well-known fact that every countable set has Hausdorff dimension 0. For smaller resource bounds Δ, Lemma 4.15 has consequences of the following sort.

COROLLARY 4.16. *For every $k \in \mathbb{N}$,*

$$\dim(\mathrm{DTIME}(2^{kn}) \mid \mathrm{E}) = 0$$

and

$$\dim(\mathrm{DTIME}(2^{n^k}) \mid \mathrm{E}_2) = 0.$$

Analogous results hold in ESPACE, REC, etc. As noted above, every countable class of languages has Hausdorff dimension 0. In contrast, we now show that, even for countable resource bounds Δ, the Δ-dimension of $R(\Delta)$ is 1. This result, which is analogous to the measure conservation theorem of [9], endows the classes $R(\Delta)$ with

internal dimensional structure. (In fact, by Observation 4.2, this result follows from the measure conservation theorem, but we prove it directly here.)

THEOREM 4.17. $\dim(R(\Delta) \mid R(\Delta)) = \dim_\Delta(R(\Delta)) = 1$.

Proof. By definition, $\dim(R(\Delta) \mid R(\Delta)) = \dim_\Delta(R(\Delta))$, and by Observation 4.2, $\dim_\Delta(R(\Delta)) \leq 1$, so it suffices to show that $\dim_\Delta(R(\Delta)) \geq 1$. For this, fix an arbitrary $s \in [0,1)$ such that 2^s is rational. By Corollary 4.6, it suffices to show that $s \notin \mathcal{G}_\Delta(R(\Delta))$. For this, let $d \in \Delta$ be an exact s-gale. By the exact computation lemma, it suffices to show that $R(\Delta) \not\subseteq S^\infty[d]$. We do this by defining a constructor $\delta \in \Delta$ such that $R(\delta) \notin S^\infty[d]$.

For each $w \in \{0,1\}^*$, let

$$\delta(w) = w[\![d(w0) > d(w1)]\!].$$

Then δ is a constructor, and it is clear that $\delta \in \Delta$ (because $d \in \Delta$). The definition of δ ensures that for all $w \in \{0,1\}^*$,

$$d(\delta(w)) = \min\{d(w0), d(w1)\}.$$

It follows by Corollary 3.4 that for all $w \in \{0,1\}^*$, $d(\delta(w)) \leq 2^{s-1} d(w)$. By induction, this implies that for all $n \in \mathbb{N}$, $d(\delta^n(\lambda)) \leq 2^{(s-1)n} d(\lambda)$. Since $s \in [0,1)$, we then have

$$\lim_{n\to\infty} d(R(\delta)[0 \ldots n-1]) = \lim_{n\to\infty} d(\delta^n(\lambda)) = 0,$$

whence $R(\delta) \notin S^\infty[d]$. □

We now give an example in which we calculate the resource-bounded dimension of a simple set of languages.

PROPOSITION 4.18. *Given $l \in \mathbb{Z}^+$ and $\emptyset \neq S \subseteq \{0,1\}^l$, let*

$$X = \{A \in \mathbf{C} \mid (\forall k) A[kl..(k+1)l-1] \in S\}.$$

Then

$$\dim(X \mid \mathrm{E}) = \dim_{\mathrm{p}}(X) = \frac{\log|S|}{l}.$$

Proof. Since $\dim(X \mid \mathrm{E}) \leq \dim_{\mathrm{p}}(X)$, it suffices to show that $\dim_{\mathrm{p}}(X) \leq \frac{\log|S|}{l}$ and $\dim(X \mid \mathrm{E}) \geq \frac{\log|S|}{l}$.

To see that $\dim_{\mathrm{p}}(X) \leq \frac{\log|S|}{l}$, let $s > \frac{\log|S|}{l}$ be such that 2^s is rational. Define a function $d : \{0,1\}^* \longrightarrow [0,\infty)$ inductively as follows. Let $d(\lambda) = 1$. If $d(w)$ has been defined, where $|w|$ is a multiple of l, and if $0 < |u| \leq l$, then let $d(wu) = 2^{s|u|}\rho(u)d(w)$, where

$$\rho(u) = \frac{|\{v \in S \mid u \sqsubseteq v\}|}{|S|}.$$

It is clear that d is exactly p-computable, and it is routine to verify that d is an s-gale. The definition of d implies that if $|w|$ is a multiple of l and $u \in S$, then

$$d(wu) = 2^{sl}\frac{1}{|S|}d(w) = 2^{\varepsilon}d(w),$$

where $\varepsilon = sl - \log|S| > 0$ by our choice of s. It follows inductively that if $A \in X$, then for all $k \in \mathbb{N}$,

$$d(A[0 \ldots kl-1]) = 2^{\varepsilon k},$$

whence $A \in S^\infty[d]$. Thus $X \subseteq S^\infty[d]$. We have now established that $\dim_{\mathrm{p}}(X) \leq s$. Since this holds for a dense set of $s > \frac{\log|S|}{l}$, it follows that $\dim_{\mathrm{p}}(X) \leq \frac{\log|S|}{l}$.

To see that $\dim(X \mid \mathrm{E}) \geq \frac{\log|S|}{l}$, let $0 \leq s < \frac{\log|S|}{l}$ be such that 2^s is rational, and let d be an exactly p-computable s-gale. By the exact computation lemma, it suffices to show that $X \cap \mathrm{E} \not\subseteq S^\infty[d]$. Define a constructor δ as follows. If $|w|$ is a multiple of l, then $\delta(w) = wu$, where u is the lexicographically first element of S such that for all $v \in S$, $d(wu) \leq d(wv)$. If $|w|$ is not a multiple of l, then $\delta(w) = w0$. Since δ is exactly p-computable, it is clear that $\delta \in \mathrm{p}$. It is then easy to see that $R(\delta) \in X \cap R(p) = X \cap \mathrm{E}$. We finish the proof by showing that $R(\delta) \notin S^\infty[d]$.

For any string w, Corollary 3.4 (with $\alpha = \frac{2^l}{|S|}$) tells us that there are fewer than $|S|$ strings $u \in \{0,1\}^l$ for which $d(wu) > \frac{2^{sl}}{|S|}d(w)$. This implies that for all w such that $|w|$ is a multiple of l, $d(\delta(w)) \leq \frac{2^{sl}}{|S|}d(w)$. Thus for all $k \in \mathbb{N}$,

$$d(R(\delta)[0 \ldots kl-1]) \leq \left(\frac{2^{sl}}{|S|}\right)^k d(\lambda) = 2^{-\varepsilon k} d(\lambda),$$

where $\varepsilon = \log|S| - sl > 0$. It follows by Corollary 3.5 that for all $n \in \mathbb{N}$,

$$d(R(\delta)[0 \ldots n-1]) \leq 2^{sl - \varepsilon\lfloor \frac{n}{l} \rfloor} d(\lambda),$$

whence $R(\delta) \notin S^\infty[d]$. □

We conclude this section by mentioning some relevant earlier work relating martingales and supermartingales to computable dimension. Schnorr [14, 15] defined a martingale d to have *exponential order* on a sequence (equivalently, language) S if

$$\limsup_{n\to\infty} \frac{\log d(S[0..n-1])}{n} > 0 \tag{4.1}$$

and proved that no computable martingale can have exponential order on a Church-stochastic sequence. Terwijn [18] has noted that (4.1) is equivalent to the existence of an $s < 1$ for which the s-gale $d^{(s)}(w) = 2^{(s-1)|w|}d(w)$ succeeds on S. Thus Schnorr's result says that $\dim_{\mathrm{comp}}(\{S\}) = 1$ for every Church-stochastic sequence S.

Ryabko [13] and Staiger [17] defined the *exponent of increase* $\lambda_d(S)$ of a martingale d on a sequence S to be the left-hand side of (4.1). (We are using Staiger's notation here.) Both papers paid particular attention to the quantity

$$\lambda(S) = \sup\{\lambda_d(S) \mid d \text{ is a computable martingale}\}. \tag{4.2}$$

By Terwijn's above-mentioned observation, $1 - \lambda(S)$ is precisely $\dim_{\mathrm{comp}}(\{S\})$. The reader is referred to these papers for interesting results relating $\lambda(S)$—and hence computable dimension—to Kolmogorov complexity and Hausdorff dimension.

5. Dimension and frequency in exponential time. In this section we show that for each $0 \leq s \leq 1$ there is a natural set X that has dimension s in each of the exponential time complexity classes E and E_2. This set X consists of those languages that asymptotically contain at most α of all strings, where $0 \leq \alpha \leq \frac{1}{2}$ and $\mathcal{H}(\alpha) = s$. We now define this set more completely.

For each nonempty string $w \in \{0,1\}^+$, let

$$\mathrm{freq}(w) = \frac{\#(1,w)}{|w|},$$

where $\#(1, w)$ is the number of 1's in w. For each $A \in \mathbf{C}$ and $n \in \mathbb{Z}^+$, let

$$\text{freq}_A(n) = \text{freq}(A[0..n-1]).$$

That is, $\text{freq}_A(n)$ is the fraction of the first n strings in $\{0,1\}^*$ that are elements of A. For $\alpha \in [0,1]$, define the sets

$$\text{FREQ}(\alpha) = \{A \in \mathbf{C} \mid \lim_{n\to\infty} \text{freq}_A(n) = \alpha\},$$

$$\text{FREQ}(\leq \alpha) = \{A \in \mathbf{C} \mid \liminf_{n\to\infty} \text{freq}_A(n) \leq \alpha\}.$$

The set $\text{FREQ}(\leq \alpha)$ is the set X promised in the preceding paragraph.

Our results use the binary entropy function

$$\mathcal{H} : [0,1] \longrightarrow [0,1]$$

$$\mathcal{H}(\alpha) = \alpha \log \frac{1}{\alpha} + (1-\alpha) \log \frac{1}{1-\alpha}.$$

(The values $\mathcal{H}(0)$ and $\mathcal{H}(1)$ are both 0, so that $\mathcal{H}$ is continuous on [0, 1].) Our proofs use the "weighted entropy" function

$$h : (0,1) \times (0,1) \longrightarrow \mathbb{R}$$

$$h(x,y) = x \log \frac{1}{y} + (1-x) \log \frac{1}{1-y}.$$

For fixed x, $h(x,y)$ takes its minimum value $\mathcal{H}(x)$ at $y = x$ and strictly increases as y moves away from x.

We first show that $\mathcal{H}(\alpha)$ is an upper bound on the p-dimension of $\text{FREQ}(\leq \alpha)$.

LEMMA 5.1. *For all* $\alpha \in [0, \frac{1}{2}]$, $\dim_{\text{p}}(\text{FREQ}(\leq \alpha)) \leq \mathcal{H}(\alpha)$.

Proof. Let $0 \leq \alpha \leq \frac{1}{2}$. Let $s > \mathcal{H}(\alpha)$ be such that 2^s is rational, and let $\varepsilon = \frac{s - \mathcal{H}(\alpha)}{2}$. Fix $\delta > 0$ such that for all $x, y \in [\alpha-\delta, \alpha+\delta] \cap (0,1)$, $|h(x,y) - \mathcal{H}(\alpha)| < \varepsilon$, and let $y \in [\alpha - \delta, \alpha + \delta] \cap (0, \frac{1}{2}]$ be a rational number. Define

$$d : \{0,1\}^* \longrightarrow \mathbb{Q} \cap [0, \infty)$$

$$d(\lambda) = 1,$$

$$d(w0) = (1-y)2^s d(w),$$

$$d(w1) = y2^s d(w).$$

It is clear that d is an exactly p-computable s-gale.

To see that $\mathrm{FREQ}(\leq \alpha) \subseteq S^\infty[d]$, let $A \in \mathrm{FREQ}(\leq \alpha)$. Then there exists an infinite set $J \subseteq \mathbb{N}$ such that for all $n \in J$, $\mathrm{freq}_A(n) \leq \alpha + \delta$. It follows that for all $n \in J$, if we write $w_n = A[0 \ldots n-1]$, then

$$\begin{aligned} d(w_n) &= y^{\#(1,w_n)}(1-y)^{\#(0,w_n)}2^{sn} \\ &= \left[y^{\mathrm{freq}_A(n)}(1-y)^{1-\mathrm{freq}_A(n)}2^s\right]^n \\ &\geq \left[y^{\alpha+\delta}(1-y)^{1-(\alpha+\delta)}2^s\right]^n \\ &= \left[2^{s-h(\alpha+\delta,y)}\right]^n \\ &\geq \left[2^{s-\mathcal{H}(\alpha)-\varepsilon}\right]^n \\ &= 2^{\varepsilon n} \ . \end{aligned}$$

Thus $A \in S^\infty[d]$.

We have now shown that $\dim_{\mathrm{p}}(\mathrm{FREQ}(\leq \alpha)) \leq s$. Since this holds for all $s > \mathcal{H}(\alpha)$ such that 2^s is rational, it follows that $\dim_{\mathrm{p}}(\mathrm{FREQ}(\leq \alpha)) \leq \mathcal{H}(\alpha)$. □

We next show that $\mathcal{H}(\alpha)$ is a lower bound on the dimension of $\mathrm{FREQ}(\alpha)$ in $R(\Delta)$. In general, this does not hold for arbitrary $\alpha \in [0,1]$, since if $R(\Delta)$ is countable, there can be only countably many α for which $\mathrm{FREQ}(\alpha) \cap R(\Delta) \neq \emptyset$. However, it does hold for all Δ-computable real numbers $\alpha \in [0,1]$.

LEMMA 5.2. *For all Δ-computable $\alpha \in [0,1]$, $\dim(\mathrm{FREQ}(\alpha) \mid R(\Delta)) \geq \mathcal{H}(\alpha)$.*

Proof. The inequality holds trivially for $\alpha \in \{0,1\}$, so assume that $\alpha \in (0,1)$ is Δ-computable. Fix $s \in (0, \mathcal{H}(\alpha))$ such 2^s is rational, and let d be an exactly Δ-computable s-gale. By the exact computation lemma, it suffices to show that $\mathrm{FREQ}(\alpha) \cap R(\Delta) \not\subseteq S^\infty[d]$.

Although the details are a bit involved, the idea of the proof is simple. We want to define a constructor $\delta \in \Delta$ such that $R(\delta) \in \mathrm{FREQ}(\alpha) - S^\infty[d]$. Given a string w of length n, δ computes a rational approximation $\frac{k(n)}{m(n)}$ of α and extends w by a string u of length $m(n)$ containing exactly $k(n)$ 1's and having the property that d makes no progress along u. Such a string u exists because $s < \mathcal{H}(\frac{k(n)}{m(n)})$, and it can be found within the resource bound Δ because $m(n)$ is logarithmic in n. On the other hand, $m(n) \to \infty$ as $n \to \infty$, so $R(\delta) \in \mathrm{FREQ}(\alpha)$, and d makes no progress along $R(\delta)$, so $R(\delta) \notin S^\infty[d]$. We now develop this idea.

Since α is Δ-computable, there is a function $\hat{\alpha} : \mathbb{N} \to \mathbb{Q} \cap (0,1)$ such that $\hat{\alpha} \in \Delta$ and for all $r \in \mathbb{N}$, $|\hat{\alpha}(r) - \alpha| \leq 2^{-r}$. Fix $\epsilon > 0$ such that

$$\epsilon \leq \alpha, \quad \epsilon \leq 1 - \alpha, \tag{5.1}$$

and for all $x \in [0,1]$,

$$|x - \alpha| < \epsilon \Rightarrow |\mathcal{H}(x) - \mathcal{H}(\alpha)| < \frac{\mathcal{H}(\alpha) - s}{2}. \tag{5.2}$$

Choose a positive integer c satisfying the conditions

$$c > 2^{1+\frac{2}{\epsilon}}, \tag{5.3}$$

$$c > 1 + \log \frac{1}{\epsilon}, \tag{5.4}$$

$$\frac{\log c}{\log \log c} > \frac{4}{\mathcal{H}(\alpha) - s}. \tag{5.5}$$

For each $n \in \mathbb{N}$, let

$$m(n) = \lfloor \log(n+c) \rfloor,$$
$$k(n) = \lfloor \hat{\alpha}(m(n)+c)m(n) \rfloor.$$

We first show that $\frac{k(n)}{m(n)}$ is a useful approximation of α. By (5.3),

$$m(n) \geq \lfloor \log c \rfloor \geq \frac{2}{\epsilon} \tag{5.6}$$

for all $n \in \mathbb{N}$. In particular, $m(n)$ is always positive. By (5.4) and (5.1),

$$\begin{aligned} \hat{\alpha}(m(n)+c) &\geq \alpha - 2^{-c} \geq \alpha - 2^{-1+\log \epsilon} \\ &= \alpha - \frac{\epsilon}{2} \geq \frac{\alpha}{2} \end{aligned} \tag{5.7}$$

for all $n \in \mathbb{N}$. It follows from (5.6), (5.7), and (5.1) that

$$k(n) \geq \left\lfloor \frac{2}{\epsilon} \frac{\alpha}{2} \right\rfloor > 0 \tag{5.8}$$

for all $n \in \mathbb{N}$. Also by (5.4) and (5.1),

$$\begin{aligned} \hat{\alpha}(m(n)+c) &\leq \alpha + 2^{-c} < \alpha + 2^{-1+\log \epsilon} \\ &\leq \alpha + 2^{-1+\log(1-\alpha)} = \frac{1+\alpha}{2} < 1, \end{aligned}$$

so

$$k(n) = \lfloor \hat{\alpha}(m(n)+c)m(n) \rfloor < m(n) \tag{5.9}$$

for all $n \in \mathbb{N}$. By (5.8) and (5.9), we now have

$$0 < \frac{k(n)}{m(n)} < 1 \tag{5.10}$$

for all $n \in \mathbb{N}$. In fact,

$$\frac{k(n)}{m(n)} = \frac{\lfloor \hat{\alpha}(m(n)+c)m(n) \rfloor}{m(n)},$$

so

$$\hat{\alpha}(m(n)+c) - \frac{1}{m(n)} < \frac{k(n)}{m(n)} \leq \hat{\alpha}(m(n)+c),$$

and thus

$$\left| \frac{k(n)}{m(n)} - \alpha \right| \leq \frac{1}{m(n)} + 2^{-(m(n)+c)} \tag{5.11}$$

for all $n \in \mathbb{N}$. It follows by (5.4) and (5.6) that

$$\left| \frac{k(n)}{m(n)} - \alpha \right| \leq \epsilon,$$

whence by (5.2),

$$\left|\mathcal{H}\left(\frac{k(n)}{m(n)}\right) - \mathcal{H}(\alpha)\right| < \frac{\mathcal{H}(\alpha) - s}{2} \tag{5.12}$$

for all $n \in \mathbb{N}$. This is the first sense in which $\frac{k(n)}{m(n)}$ is a useful approximation of α.

For each $n \in \mathbb{N}$, define the set

$$B_n = \left\{ u \in \{0,1\}^{m(n)} \,\middle|\, \#(1,u) = k(n) \right\}.$$

Using (5.10) and the well-known approximation $e\left(\frac{N}{e}\right)^N < N! < eN\left(\frac{N}{e}\right)^N$, valid for all $N \geq 1$, it is easy to see that

$$\begin{aligned} |B_n| = \binom{m(n)}{k(n)} &> \frac{1}{ek(n)(m(n)-k(n))} 2^{m(n)\mathcal{H}\left(\frac{k(n)}{m(n)}\right)} \\ &\geq \frac{4}{em(n)^2} 2^{m(n)\mathcal{H}\left(\frac{k(n)}{m(n)}\right)} \\ &> 2^{m(n)\mathcal{H}\left(\frac{k(n)}{m(n)}\right) - 2\log m(n)} \end{aligned}$$

for all $n \in \mathbb{N}$. It follows by (5.12) that

$$\begin{aligned} |B_n| &> 2^{m(n)\left[\mathcal{H}(\alpha) - \frac{\mathcal{H}(\alpha)-s}{2}\right] - 2\log m(n)} \\ &= 2^{m(n)\frac{\mathcal{H}(\alpha)+s}{2} - 2\log m(n)} \end{aligned}$$

for all $n \in \mathbb{N}$. Now (5.5) and the monotonicity of $\frac{\log x}{\log\log x}$ tell us that

$$m(n)\frac{\mathcal{H}(\alpha) - s}{2} \geq 2\log m(n),$$

so it follows that

$$|B_n| > 2^{m(n)\left[\frac{\mathcal{H}(\alpha)+s}{2} - \frac{\mathcal{H}(\alpha-s)}{2}\right]} = 2^{sm(n)} \tag{5.13}$$

for all $n \in \mathbb{N}$.

Corollary 3.4 tells us that for each $w \in \{0,1\}^*$ there are fewer than $2^{sm(n)}$ strings $u \in \{0,1\}^{m(n)}$ for which $\max_{v \sqsubseteq u} d(wv) > d(w)$. It follows by (5.13) that the function

$$\delta : \{0,1\}^* \to \{0,1\}^*$$

$$\begin{aligned} \delta(w) = wu, \quad &\text{where } u \text{ is the lexicographically} \\ &\text{first element of } B_{|w|} \text{ such that} \\ &d(wv) \leq d(w) \text{ for all } v \sqsubseteq u, \end{aligned}$$

is a well-defined constructor. Also, since $d \in \Delta$ and $|\{0,1\}^{m(|w|)}| \leq |w| + c$, it is clear that $\delta \in \Delta$. To conclude the proof, then, it suffices to show that $R(\delta) \in \mathrm{FREQ}(\alpha) - S^\infty[d]$.

To see that $R(\delta) \in \mathrm{FREQ}(\alpha)$, define a sequence $n_0, n_1, \ldots$ by the recursion

$$n_0 = 0, \quad n_{i+1} = n_i + m(n_i),$$

and note that $R(\delta)$ is of the form

$$R(\delta) = u_0 u_1 u_2 \cdots,$$

where $|u_i| = m(n_i)$ and $\text{freq}(u_i) = \frac{k(n_i)}{m(n_i)}$ for all $i \in \mathbb{N}$. Also, by (5.11) we have

$$\left| \frac{k(n)}{m(n)} - \alpha \right| \leq \frac{2}{m(n)}$$

for all $n \in \mathbb{N}$. Since $m(n) \to \infty$ as $n \to \infty$, it follows that

$$\lim_{i \to \infty} \text{freq}(u_i) = \lim_{i \to \infty} \frac{k(n_i)}{m(n_i)} = \alpha.$$

(This is the second sense in which $\frac{k(n)}{m(n)}$ is a useful approximation of α.) Since $|u_i|$ is nondecreasing, this implies that $R(\delta) \in \text{FREQ}(\alpha)$.

The definition of δ implies that $d(w) \leq d(\lambda)$ for all $w \sqsubseteq R(\delta)$, so $R(\delta) \notin S^\infty[d]$. □

From these two lemmas we get the main result of this section.

THEOREM 5.3.

1. *For all Δ-computable $\alpha \in [0,1]$,* $\dim(\text{FREQ}(\alpha) \mid R(\Delta)) = \mathcal{H}(\alpha)$.
2. *For all $\alpha \in [0, \frac{1}{2}]$,* $\dim(\text{FREQ}(\leq \alpha) | R(\Delta)) = \mathcal{H}(\alpha)$.

Proof.

1. This follows immediately from Lemmas 5.1 and 5.2 and Observation 4.2.
2. Let $\alpha \in [0, \frac{1}{2}]$. By Lemma 5.1 and Observation 4.2,

$$\begin{aligned} &\dim(\text{FREQ}(\leq \alpha) \mid R(\Delta)) \\ &= \dim_\Delta(\text{FREQ}(\leq \alpha) \cap R(\Delta)) \\ &\leq \dim_\Delta(\text{FREQ}(\leq \alpha)) \\ &\leq \dim_{\text{p}}(\text{FREQ}(\leq \alpha)) \\ &\leq \mathcal{H}(\alpha). \end{aligned}$$

For the reverse inequality, let α' be an arbitrary rational such that $\alpha' \leq \alpha$. Then by Lemma 5.2 and Observation 4.2,

$$\begin{aligned} &\dim(\text{FREQ}(\leq \alpha) \mid R(\Delta)) \\ &\geq \dim(\text{FREQ}(\alpha') \cap R(\Delta)) \\ &\geq \mathcal{H}(\alpha'). \end{aligned}$$

Since this holds for all rational $\alpha' \leq \alpha$ and $\mathcal{H}$ is continuous, it follows that

$$\dim(\text{FREQ}(\leq \alpha) \mid R(\Delta)) \geq \mathcal{H}(\alpha). \quad \square$$

The case $\Delta =$ all of Theorem 5.3 says simply that the classical Hausdorff dimensions of $\text{FREQ}(\alpha)$ and $\text{FREQ}(\leq \alpha)$ are both $\mathcal{H}(\alpha)$. This was proven in 1949 by Eggleston [5, 2]. The proof here yields a new proof, using gales, of this classical result. However, it is complexity-theoretic results of the following kind that are of interest in this paper.

COROLLARY 5.4.

1. *For all p-computable reals* $\alpha \in [0,1]$,

$$\dim(\mathrm{FREQ}(\alpha) \mid \mathrm{E}) = \dim(\mathrm{FREQ}(\alpha) \mid \mathrm{E}_2) = \mathcal{H}(\alpha).$$

2. *For all* $\alpha \in [0, \frac{1}{2}]$,

$$\dim(\mathrm{FREQ}(\leq \alpha) \mid \mathrm{E}) = \dim(\mathrm{FREQ}(\leq \alpha) \mid \mathrm{E}_2) = \mathcal{H}(\alpha).$$

Since $\mathcal{H}(0) = 0$, $\mathcal{H}(\frac{1}{2}) = 1$, and $\mathcal{H}$ is continuous, part 2 of the corollary implies that for every $s \in [0,1]$ there exists $\alpha \in [0, \frac{1}{2}]$ such that

$$\dim(\mathrm{FREQ}(\leq \alpha) \mid \mathrm{E}) = \dim(\mathrm{FREQ}(\leq \alpha) \mid \mathrm{E}_2) = s.$$

6. Dimension and circuit size in exponential space. We now examine the dimensions of Boolean circuit-size complexity classes in the complexity class ESPACE.

Our Boolean circuit model and terminology are standard. Details may be found in [9], but our result is not sensitive to minor details of the model. The *circuit-size complexity* of a language $A \subseteq \{0,1\}^*$ is the function $CS_A : \mathbb{N} \longrightarrow \mathbb{N}$, where $CS_A(n)$ is the number of gates in the smallest n-input Boolean circuit that decides $A \cap \{0,1\}^n$. For each function $f : \mathbb{N} \longrightarrow \mathbb{N}$, we define the circuit-size complexity class

$$\mathrm{SIZE}(f) = \{A \in \mathbf{C} \mid (\forall^\infty n) CS_A(n) \leq f(n)\}.$$

Shannon [16] showed (essentially) that $\mathrm{SIZE}(\alpha \frac{2^n}{n})$ has measure 0 in $\mathbf{C}$ for all $\alpha < 1$, and Lutz [9] showed that $\mathrm{SIZE}(\alpha \frac{2^n}{n})$ also has measure 0 in ESPACE for all $\alpha < 1$. We now use resource-bounded dimension to give a quantitative refinement of these results.

Assume that all n-input Boolean circuits are enumerated in a canonical order in which all circuits of size (number of gates) t precede all those of size $t+1$ for all $t \in \mathbb{Z}^+$. Call a circuit in this enumeration *novel* if there is no previous circuit in the enumeration that decides the same subset of $\{0,1\}^n$. For each $n \in \mathbb{N}$ and $t \in \mathbb{Z}^+$, let $N(n,t)$ be the number of novel n-input Boolean circuits with at most t gates. The following specific bound on $N(n,t)$ was proven in [9], but as noted there, the idea is essentially due to Shannon [16].

LEMMA 6.1. *For all* $n \in \mathbb{N}$ *and* $t > n$, $N(n,t) \leq (48et)^t$.

LEMMA 6.2. *For every real* $\alpha \in [0,1]$, $\dim_{\mathrm{pspace}}(\mathrm{SIZE}(\alpha \frac{2^n}{n})) \leq \alpha$.

Proof. Let $\alpha \in (0,1]$, and let $s > \alpha$ be such that 2^s is rational. Define $d : \{0,1\}^* \longrightarrow [0,\infty)$ inductively as follows:

(i) Let $d(\lambda) = 1$.

(ii) Assume that $d(w)$ has been defined, where $|w| = 2^n - 1$ for some $n \in \mathbb{N}$. For each u with $0 < |u| \leq 2^n$, define $d(wu) = 2^{s|u|}\rho(u)d(w)$, where

$$\rho(u) = \frac{N(n,t,u)}{N(n,t)}$$

and $N(n,t,u)$ is the number of novel n-input Boolean circuits with at most t gates that decide the sets $B \subseteq \{0,1\}^n$ whose first $|u|$ decisions are the bits of u. It is easy to check that d is an exactly pspace-computable s-gale. The definition of d implies that if $|w| = 2^n - 1$ and u is the characteristic string

of a set $B \subseteq \{0,1\}^n$ that can be decided by a circuit with at most $\alpha\frac{2^n}{n}$ gates, then for sufficiently large n,

$$\begin{aligned} d(wu) &= 2^{s2^n}\frac{1}{N(n,\alpha\frac{2^n}{n})}d(w) \\ &= 2^{s2^n-\alpha\frac{2^n}{n}[\log 48e+\log\alpha+n-\log n]}d(w) \\ &= 2^{(s-\alpha)2^n+\alpha\frac{2^n}{n}[\log n-\log 48e-\log\alpha]}d(w) \\ &\geq 2^{(s-\alpha)2^n}d(w). \end{aligned}$$

Since $s-\alpha>0$, this implies that $\mathrm{SIZE}(\alpha\frac{2^n}{n}) \subseteq S^\infty[d]$. □

LEMMA 6.3. *If $0<\beta<\alpha\leq 1$, then for all sufficiently large n there are at least $2^{\beta 2^n}$ different sets $B\subseteq\{0,1\}^n$ that are decided by Boolean circuits of fewer than $\alpha\frac{2^n}{n}$ gates.*

Proof. Let $m=n+\log\beta$. Then there are $2^{2^m}=2^{\beta 2^n}$ different sets $C\subseteq\{0,1\}^m$. If we let $\varepsilon=\frac{\alpha-\beta}{2\alpha}$, so that $\beta=\alpha(1-2\varepsilon)$, then for all sufficiently large n, Lupanov [8] has shown that each of these sets is decided by a circuit of at most $\frac{2^m}{m}(1+\varepsilon)$ gates. Now for sufficiently large n,

$$\begin{aligned} \frac{2^m}{m} &= \frac{\beta 2^n}{n+\log\beta} \\ &= \alpha\frac{2^n}{n}(1-2\varepsilon)\left(\frac{n}{n+\log\beta}\right) \\ &< \alpha\frac{2^n}{n}(1-\varepsilon), \end{aligned}$$

so

$$\frac{2^m}{m}(1+\varepsilon) < \alpha\frac{2^n}{n}(1-\varepsilon^2) < \alpha\frac{2^n}{n}.$$

Thus, for each $C\subseteq\{0,1\}^m$, if we let $B_C=\{w0^{n-m}\mid w\in C\}$, then B_C is decided by a Boolean circuit of fewer than $\alpha\frac{2^n}{n}$ gates. □

Recall that in section 2 we defined constructors and showed that $R(\mathrm{pspace}) = \mathrm{ESPACE}$.

LEMMA 6.4. *For every real $\alpha\in[0,1]$, $\dim_H(\mathrm{SIZE}(\alpha\frac{2^n}{n}))\geq\alpha$ and*

$$\dim\left(\mathrm{SIZE}\left(\alpha\frac{2^n}{n}\right)\,\middle|\,\mathrm{ESPACE}\right)\geq\alpha.$$

Proof. This is clear if $\alpha=0$, so assume that $\alpha\in(0,1]$. Let $0<s<\alpha$ with 2^s rational, and let d be an arbitrary s-gale. Define a constructor $\delta:\{0,1\}^*\to\{0,1\}^*$ as follows. If $|w|$ is not of the form 2^n-1, then $\delta(w)=w0$. If $|w|$ is of the form 2^n-1 (i.e., w is the characteristic string of a subset of $\{0,1\}^{<n}$), then $\delta(w)=wu$, where u is the first string in $\{0,1\}^{2^n}$ that minimizes $\max_{v\sqsubseteq u}d(wv)$, subject to the constraint that u is the characteristic string of a set $B\subseteq\{0,1\}^n$ that is decided by a Boolean circuit with fewer than $\alpha\frac{2^n}{n}$ gates. It is clear that $R(\delta)\in\mathrm{SIZE}(\alpha\frac{2^n}{n})$.

Fix β such that $s<\beta<\alpha$. By Lemma 6.3, for all sufficiently large n there are at least $2^{\beta 2^n}$ different sets $B\subseteq\{0,1\}^n$ that are decided by Boolean circuits of fewer than $\alpha\frac{2^n}{n}$ gates. By Corollary 3.4, for all w such that $|w|=2^n-1$, there are fewer than

$2^{\beta 2^n}$ strings $u \in \{0,1\}^{2^n}$ such that $\max_{v \sqsubseteq u} d(wv) > 2^{|v|(s-\beta)} d(w)$. Taken together, these last two sentences imply that for all sufficiently large n and w with $|w| = 2^n - 1$,

$$\max_{wv \sqsubseteq \delta(w)} d(wv) \leq 2^{|v|(s-\beta)} d(w) \leq d(w).$$

It follows from this that $R(\delta) \notin S^\infty[d]$.

We now have that $R(\delta) \in \mathrm{SIZE}(\alpha \frac{2^n}{n}) - S^\infty[d]$, whence $\mathrm{SIZE}(\alpha \frac{2^n}{n}) \not\subseteq S^\infty[d]$. Since d is arbitrary here, this shows that $\dim_H(\mathrm{SIZE}(\alpha \frac{2^n}{n})) \geq \alpha$.

Now let d be as above, and assume further that d is exactly pspace-computable. Then the constructor δ defined above is in pspace, so $R(\delta) \in R(\mathrm{pspace}) = \mathrm{ESPACE}$, and thus we have $\mathrm{SIZE}(\alpha \frac{2^n}{n}) \cap \mathrm{ESPACE} \not\subseteq S^\infty[d]$. It follows by Lemma 4.8 that $\dim(\mathrm{SIZE}(\alpha \frac{2^n}{n}) | \mathrm{ESPACE}) \geq \alpha$. □

THEOREM 6.5. *For every real $\alpha \in [0,1]$,*

$$\dim\left(\mathrm{SIZE}\left(\alpha \frac{2^n}{n}\right) \middle| \mathrm{ESPACE}\right) = \dim_H\left(\mathrm{SIZE}\left(\alpha \frac{2^n}{n}\right)\right) = \alpha.$$

Proof. This follows immediately from Lemma 6.2, Lemma 6.4, and Observation 4.2. □

We note that for any $\alpha < 1$, Lutz [9] has shown that the class $\mathrm{SIZE}(\frac{2^n}{n}(1 + \frac{\alpha \log n}{n}))$ has measure 0 in ESPACE and in $\mathbf{C}$. By the above results, this class is thus a natural example of a set that, in both ESPACE and $\mathbf{C}$, has dimension 1 but measure 0.

Acknowledgments. I am very grateful to Elvira Mayordomo, John Hitchcock, and Dan Mauldin for useful discussions.

REFERENCES

[1] K. AMBOS-SPIES AND E. MAYORDOMO, *Resource-bounded measure and randomness,* in Complexity, Logic and Recursion Theory, A. Sorbi, ed., Lecture Notes in Pure and Appl. Math., Marcel Dekker, New York, 1997, pp. 1–47.

[2] P. BILLINGSLEY, *Ergodic Theory and Information*, John Wiley, New York, 1965.

[3] H. BUHRMAN AND L. TORENVLIET, *Complete sets and structure in subrecursive classes,* in Proceedings of Logic Colloquium '96, Springer-Verlag, Berlin, 1998, pp. 45–78.

[4] J. J. DAI, *A stronger Kolmogorov zero-one law for resource-bounded measure,* Theoret. Comput. Sci., 292 (2003), pp. 723–732.

[5] H. G. EGGLESTON, *The fractional dimension of a set defined by decimal properties,* Quart. J. Math. Oxford Ser., 20 (1949), pp. 31–36.

[6] K. FALCONER, *The Geometry of Fractal Sets*, Cambridge University Press, Cambridge, UK, 1985.

[7] F. HAUSDORFF, *Dimension und äusseres mass,* Math. Ann., 79 (1919), pp. 157–179.

[8] O. B. LUPANOV, *On the synthesis of contact circuits,* Dokl. Akad. Nauk SSSR (N.S.), 119 (1958), pp. 23–26.

[9] J. H. LUTZ, *Almost everywhere high nonuniform complexity,* J. Comput. System Sci., 44 (1992), pp. 220–258.

[10] J. H. LUTZ, *The quantitative structure of exponential time,* in Complexity Theory Retrospective II, L. A. Hemaspaandra and A. L. Selman, eds., Springer-Verlag, New York, 1997, pp. 225–254.

[11] J. H. LUTZ, *Resource-bounded measure,* in Proceedings of the 13th IEEE Conference on Computational Complexity, IEEE Computer Society Press, New York, 1998, pp. 236–248.

[12] J. H. LUTZ AND E. MAYORDOMO, *Twelve problems in resource-bounded measure,* Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 68 (1999), pp. 64–80.

[13] B. YA. RYABKO, *The complexity and effectiveness of prediction problems,* J. Complexity, 10 (1994), pp. 281–295.

[14] C. P. SCHNORR, *Zufälligkeit und Wahrscheinlichkeit,* Lecture Notes in Math. 218, Springer-Verlag, Berlin, 1971.

[15] C. P. Schnorr, *A survey of the theory of random sequences,* in Basic Problems in Methodology and Linguistics, R. E. Butts and J. Hintikka, eds., D. Reidel, Boston, MA, 1977, pp. 193–210.
[16] C. E. Shannon, *The synthesis of two-terminal switching circuits,* Bell System Tech. J., 28 (1949), pp. 59–98.
[17] L. Staiger, *A tight upper bound on Kolmogorov complexity and uniformly optimal prediction,* Theory Comput. Syst., 31 (1998), pp. 215–229.
[18] S. A. Terwijn, *Personal communication*, 2000.
[19] J. Ville, *Étude Critique de la Notion de Collectif*, Gauthier–Villars, Paris, 1939.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1260–1279

THE NATURAL WORK-STEALING ALGORITHM IS STABLE*

PETRA BERENBRINK†, TOM FRIEDETZKY†, AND LESLIE ANN GOLDBERG‡

Abstract. In this paper we analyze a very simple dynamic work-stealing algorithm. In the work-generation model, there are n (work) *generators*. A *generator-allocation function* is simply a function from the n generators to the n processors. We consider a fixed, but arbitrary, distribution $\mathcal{D}$ over generator-allocation functions. During each time step of our process, a generator-allocation function h is chosen from $\mathcal{D}$, and the generators are allocated to the processors according to h. Each generator may then generate a unit-time task, which it inserts into the queue of its host processor. It generates such a task independently with probability λ. After the new tasks are generated, each processor removes one task from its queue and services it. For many choices of $\mathcal{D}$, the work-generation model allows the load to become arbitrarily imbalanced, even when $\lambda < 1$. For example, $\mathcal{D}$ could be the point distribution containing a single function h which allocates all of the generators to just one processor. For this choice of $\mathcal{D}$, the chosen processor receives around λn units of work at each step and services one. The natural work-stealing algorithm that we analyze is widely used in practical applications and works as follows. During each time step, each *empty* processor (with no work to do) sends a request to a randomly selected other processor. Any *nonempty* processor having received at least one such request in turn decides (again randomly) in favor of one of the requests. The number of tasks which are transferred from the nonempty processor to the empty one is determined by the so-called *work-stealing function* f. In particular, if a processor that accepts a request has ℓ tasks stored in its queue, then $f(\ell)$ tasks are transferred to the currently empty one. A popular work-stealing function is $f(\ell) = \lfloor \ell/2 \rfloor$, which transfers (roughly) half of the tasks. We analyze the *long-term behavior* of the system as a function of λ and f. We show that the system is *stable* for any constant generation rate $\lambda < 1$ and for a wide class of functions f. Most intuitively sensible functions are included in this class (for example, every monotonically nondecreasing function f which satisfies $0 \le f(\ell) \le \ell/2$ and $f(\ell) = \omega(1)$ as a function of ℓ is included). Furthermore, we give *upper bounds* on the average system load (as a function of f and n). Our proof techniques combine Lyapunov function arguments with domination arguments, which are needed to cope with dependency.

Key words. work-stealing, dynamic, stability

AMS subject classification. 60J20

DOI. 10.1137/S0097539701399551

1. Introduction.

Load balancing is the process of distributing load among a set of processors. There are two main approaches to distributed load balancing, namely, *sender-initiated* strategies, in which processors may decide to give away tasks, and *receiver-initiated* strategies (which are often referred to as *work-stealing*), in which processors may request extra work. In both cases, the decision to transfer tasks is typically *threshold based*. That is, it is based on having too many or too few tasks in one's own queue.

In recent years, there has been a lot of work devoted to rigorously analyzing load balancing, most of which concentrates on sender-initiated approaches or related

*Received by the editors December 12, 2001; accepted for publication (in revised form) February 13, 2003; published electronically August 6, 2003. This work was partially supported by EPSRC grant GR/M96940 "Sharper Analysis of Randomised Algorithms: A Computational Approach" and the Future and Emerging Technologies Programme of the EU under contracts IST-1999-14186 (ALCOM-FT) and IST-1999-14036 (RAND-APX). A preliminary version of this article was published in the *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science* (FOCS'01), IEEE Computer Society Press, Los Alamitos, CA, 2001.

http://www.siam.org/journals/sicomp/32-5/39955.html

†School of Computing Science, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada (petra@cs.sfu.ca, tkf@cs.sfu.ca).

‡Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK (leslie@dcs.warwick.ac.uk).

allocation processes such as *balls-into-bins games*. However, it appears that many practitioners prefer the receiver-initiated approach (work-stealing) because this approach appears to work better for their applications. The efficiency of work-stealing probably helps to explain the success of Leiserson et al.'s language `Cilk` [10], a language for multithreaded parallel programming which uses work-stealing in its kernel. There are numerous examples of practical applications of work-stealing. In [17], Feldmann, Mysliwietz, and Monien investigate the behavior of parallel MIN/MAX-tree evaluation in the context of parallel game (chess) programs employing work-stealing strategies. In [14], Decker introduces VDS (virtual data space), a load-balancing system for irregular applications that makes use of work-stealing and other strategies. In [23], Mahapatra and Dutt use work-stealing for parallel branch and bound algorithms.

Despite the practical usefulness of work-stealing, there are not many known theoretical results about its performance. Most existing theoretical work on load balancing assumes a rather well-behaved system. For instance, most work on sender-initiated load balancing uses a work-generation model in which each processor generates at most a constant number of new tasks per step. In balls-into-bins games, each ball (task) chooses its bin (processor) uniformly at random (u.a.r.), which also yields a relatively balanced system.

In this paper we analyze a simple and fully distributed work-stealing algorithm. Our work-generation model allows for an arbitrary placement of n so-called *generators* among the set of n processors. Each generator generates a new task with a certain probability λ at each time step. In the extreme case, there can be one processor being host to n generators. In this case the one processor has an expected increase of $\lambda n - 1$ tasks per step, whereas all other processors do not generate tasks at all.

Our load-balancing algorithm follows a very simple and natural work-stealing approach. At each time step, each *empty* processor sends a request to one randomly chosen other processor. Each *nonempty* processor having received at least one such request selects one of them randomly. Now each empty processor P whose request is accepted by a processor Q "steals" $f(\ell)$ tasks from Q, where ℓ denotes Q's load.

Our results are concerned mostly with the *stability* of the system. A system is said to be *unstable* if the system load (the sum of the load of all processors) grows unboundedly with time. We present both negative and positive results, depending on the work-stealing function f. First we show that if the work-stealing function is $\omega(1)$ as a function of the load (i.e., $f(\ell) = \omega(1)$ as a function of ℓ), then the system is stable (provided f is monotonically nondecreasing and satisfies $0 \le f(\ell) \le \ell/2$). This result still holds if we put an upper bound on the amount of work that can be "stolen" by a single request. That is, for an upper bound h_z which is independent of ℓ (but depends on n) and will be defined in Lemma 2, the work-stealing function defined by $f'(\ell) = \min(f(\ell), f(h_z))$ is also stable. The value h_z depends upon the function f, but it need not be very large. For the function $f(\ell) = \lfloor \ell/2 \rfloor$, the value h_z is bounded from above by a polynomial in n. See section 3 for details. Our stability results are complemented by a straightforward lower bound: The system is unstable if $f(\ell)$ is too small, for example, if $f(\ell) < \lambda n - 1$. Finally, we provide upper bounds on the system load in a stationary system (again, depending on f).

1.1. New results. Before we state our results, we introduce our model and the work-stealing algorithm.

The model. We start with a collection of n synchronized *processors*, connected by some network topology. During each time step, every processor may send a request

(requesting extra work) to at most one other processor, and any processor receiving more than one such request accepts at most one of them.

In our model, we have n *generators.* A *generator-allocation function* is a function from the n generators to the n processors. We consider a fixed, but arbitrary, distribution $\mathcal{D}$ over generator-allocation functions. During each time step of our process, a generator-allocation function h is chosen from $\mathcal{D}$, and the generators are allocated to the processors according to h. Each generator may then generate a unit-time task which it inserts into the queue of its host processor. It generates such a task independently with probability $\lambda \in [0, 1]$. After the new tasks are generated, each processor removes one task from its queue and services it. We assume constant service time for all tasks.

In the absence of a load-balancing mechanism, many choices of $\mathcal{D}$ allow the load to become arbitrarily imbalanced, even when $\lambda < 1$.

The algorithm. The work-stealing algorithm is very simple and natural. During each time step, each *empty* processor (with no work to do) sends a request to one other processor, which is chosen independently and u.a.r. Each *nonempty* processor that received at least one request selects one of these independently and u.a.r. Then each empty processor whose request was accepted "steals" tasks from the other processor.

The number of tasks which are transferred from the nonempty processor to the empty one is determined by the so-called *work-stealing function* f. In particular, if a processor that accepts a request has ℓ tasks stored in its queue, then $f(\ell)$ tasks are transferred to the currently empty one. A popular work-stealing function is $f(\ell) = \lfloor \ell/2 \rfloor$, which transfers (roughly) half of the tasks.

The results. Recall that a system is said to be stable if the system load (the sum of the load of all processors) does not grow unboundedly with time. Obviously, stability for large arrival rates is one of the most desirable features of load-balancing algorithms.

In Theorem 10 we show that, given a suitable work-stealing function, our algorithm yields a stable system for *any* constant arrival rate $\lambda < 1$ and any distribution of the generators. Most intuitively sensible work-stealing functions are suitable (for example, every monotonically nondecreasing function $f(\ell)$ which is $\omega(1)$ as a function of ℓ and satisfies $0 \le f(\ell) \le \ell/2$ is suitable). The rough requirement (for f to be suitable) is that for some finite value Φ_f (which may depend upon n) and some $z = O(\log n)$ and $T = \Theta(\log n)$, we may apply f to Φ_f z times and the resulting value is still at least $2T$. (That is, $f^z(\Phi_f) \ge 2T$.) Our stability result still holds if we put an appropriate upper bound on the amount of work that can be stolen by a single request. Details are given in section 3.

In Theorem 12 we provide upper bounds on the expected system load as well as corresponding tail bounds. The upper bounds are described in terms of Φ_f and n. For many natural work-stealing functions f, Φ_f is at most a polynomial in n, so the system-load bounds are polynomial in n. For example, Φ_f is at most a polynomial in n for the natural work-stealing function $f(\ell) = \lfloor \ell/2 \rfloor$.

Finally, in Theorem 22, we classify some work-stealing functions that do not result in a stable system. For example, the system is unstable if $f(\ell) < \lambda n - 1$.

The proofs. Since the proofs are technical, we briefly introduce the underlying idea. We model our system by a discrete-time, countable Markov chain in which states are tuples giving the number of tasks currently allocated to each processor. The standard method for determining whether such a Markov chain is ergodic (i.e., whether it

has a stationary distribution) is to find an appropriate Lyapunov function [11, 16, 25] (a potential function with an appropriate drift). Foster's theorem (see Theorem 2.2.3 of [16]) shows that the chain is ergodic if and only if there is a positive Lyapunov function which is expected to decrease by a positive amount from every state except some finite subset of the state space. For many computer science applications, it is apparently prohibitively difficult to find such a one-step Lyapunov function, even when one is known to exist. Thus, multiple-step analysis is used [21, 18, 3]. We use the multiple-step extension of Foster's theorem due to Fayolle, Malyshev, and Menshikov (see Lemma 7). The technical difficulty of our proof arises because of the lack of independence as the system evolves over multiple steps. To derive our results, we study the behavior of a different and simpler Markov chain. The new Markov chain does not dominate the original chain forever, but we show that it does dominate the original chain for a sufficiently long period of time, and this enables us to prove that the original chain is ergodic. The proof of ergodicity, together with a martingale bound of [8], gives us our bound on the stationary behavior of the chain.

1.2. Known results. Most known theoretical results on load balancing are for unconditional algorithms (which perform load balancing every few steps, regardless of the system state) or for sender-initiated approaches. First, there has been a lot of work on *static* problems, in which the number of jobs to be serviced is fixed and may even be known in advance. For these results, see [4, 34, 13, 2, 5].

In our paper, we work on *dynamic* load balancing, in which tasks are generated over time. We will now describe previous work on this problem. In [1], Adler, Berenbrink, and Schröder consider a process in which m jobs arrive in each round to n servers and each server is allowed to remove one job per round. They introduce a simple algorithm in which each job chooses between 2 random servers. They show that, provided $m \leq n/6e$, no job is likely to wait more than $O(\log \log n)$ rounds. In [12] the authors analyze several dynamic balls-into-bins games with deletion.

In [26], Mitzenmacher presents a new differential-equations approach for analyzing both static and dynamic load-balancing strategies. He demonstrates the approach by providing an analysis of the supermarket model: jobs (customers) arrive as a Poisson stream of rate λn, $\lambda < 1$, at a collection of n servers. Each customer chooses d servers independently and u.a.r. and waits for service at the one with the shortest queue. The service time of the customers is exponentially distributed with mean 1. Mitzenmacher achieves results on the expected time that a customer spends in the system. Furthermore, he shows that for any time interval of fixed length, the maximum system load is likely to be at most $\log \log n / \log d + O(1)$. In [35] Vvedenskaya, Dobrushin, and Karpelevich independently present similar results. For related results, see [27, 30, 28].

In [33], Rudolph, Slivkin-Allalouf, and Upfal present a simple distributed load-balancing strategy. They consider a work-generation model in which, at every time step, the load change of any processor due to local generation and service is bounded by some constant. The balancing strategy works as follows. Each processor stores its tasks in a local queue. Whenever a processor accesses its local queue, the processor performs a balancing operation with a probability inversely proportional to the size of its queue. The balancing operation examines the queue size of a randomly chosen processor and then equalizes their load. They show that the expected load of any processor at any point of time is within a constant factor of the average load.

In [22], Lüling and Monien use a similar work-generation model. A processor initiates a load-balancing action if its load has changed by a constant factor since

its last balancing action. They show that the expected load difference between any two processors is bounded by a constant factor. They also bound the corresponding variance.

In [6, 7] the authors introduce and investigate the performance of certain randomized load-balancing algorithms under stochastic and adversarial work-generation models. They consider two different work-generation models. In the first model, in each step, each processor generates a task with some probability $p < 1$, and then each nonempty processor services a task with probability $p(1+\epsilon)$ for $\epsilon > 0$. In the second model, each processor is allowed to change its load in each step, provided that the load is only increased or decreased by at most a fixed constant amount. With high probability, the algorithms balance the system load up to additive terms of $O(\log\log n)$ and $O((\log\log n)^2)$, respectively. In particular, in the first model, the maximum load of any processor can be upper bounded by one of these terms (depending on the algorithm), whereas in the second model, the maximum load of any processor can be upper bounded by the average load plus $O(\log\log n)$. The algorithms and analysis of [6, 7] are fundamentally different from the one considered here. In particular, their algorithms are sender-initiated, i.e., overloaded processors seek to distribute their load. Moreover, their algorithms are considerably more complicated than ours.

There is relatively little existing theoretical work on receiver-initiated approaches. The interesting thing is that this approach seems to be highly efficient in practice (much more than, say, "give-away-if-overloaded"), but there are no (or hardly any) rigorous theoretical results.

In [29], Mitzenmacher applies his differential-equations approach in order to analyze several randomized work-stealing algorithms in a dynamic setting. In contrast to our work, he assumes that every processor has a Poisson generating process with rate $\lambda < 1$. Hence, in contrast to our generation model, the load is generated in a more-or-less balanced fashion and the system is stable even without any work-stealing. Each task has an exponentially distributed service time with mean 1. He models a number of work-stealing algorithms with differential equations and compares the equations with each other in order to predict which strategies will be most successful. For each set of equations, he shows that the queue-lengths decrease more quickly than for a set of equations which models the process with no work-stealing.

In [9], Blumofe and Leiserson analyze a scheduling strategy for *strict multithreaded computations*. A multithreaded computation consists of a set of threads, each of which is a sequential ordering of unit-time tasks. During a computation, a thread can spawn other threads, which are stored in a local queue. They present a work-stealing algorithm in which every idle processor tries to steal a thread from a randomly chosen other processor. The analysis shows that the expected time to execute such a multithreaded computation on P processors is $O(T_1/P + T_\infty)$, where T_1 denotes the minimum sequential execution time of the multithreaded computation, and T_∞ denotes the minimum execution time with an infinite number of processors. Furthermore, they estimate the probability that the execution time is increased by an additional factor. In [15], Fatourou and Spirakis develop an algorithm for k-strict multithreaded computations. In this case, all data dependencies of a thread are directed to ancestors in at most k higher levels.

2. The work-stealing process. Suppose that we have n processors and n generators, which create work for the processors. Each processor keeps a queue of jobs which need to be done. The evolution of the system can be described by a Markov chain X. The state X_t after step t is a tuple $(X_t(1), \ldots, X_t(n))$ in which $X_t(i)$ rep-

resents the length of the ith processor's queue after step t. Initially, all of the queues are empty, so the start state is $(0, \ldots, 0)$.

Let $N = \{1, \ldots, n\}$. Let $\mathcal{P} = \{h \mid N \to N\}$ be the set of all generator-allocation functions. When generators are allocated according a particular function $h \in \mathcal{P}$, $h(i)$ is designated as the host of the ith generator. The Markov chain X has three parameters. $\mathcal{D}$ is an arbitrary distribution over $\mathcal{P}$. A new generator-allocation function is selected from $\mathcal{D}$ during each step of the chain. The parameter λ governs the rate at which jobs are generated—each generator creates a job during each step independently with probability λ and adds the job to the queue of its current host. Finally, the function f is the work-stealing function. In section 3, we will state the properties that f must satisfy for our analysis. Figure 1 describes the transition from state X_t to state X_{t+1}.

1. Choose the generator-allocation function h_t from $\mathcal{D}$.
2. Each generator generates a new job independently with probability λ. It adds the job to the queue of its current host. In particular, the ith processor updates the size of its queue from $X_t(i)$ to $X'_t(i)$, where $X'_t(i)$ is defined to be $X_t(i)$ plus the sum of $|h_t^{-1}(i)|$ independent Bernoulli random variables with mean λ.
3. Each processor with an empty queue chooses a request destination u.a.r. from N. Formally, $r_t(i)$ is defined to be 0 if $X'_t(i) > 0$. Otherwise, $r_t(i)$ is chosen u.a.r. from N.
4. Each processor which receives a request chooses a recipient and allocates some of its load to give away to the recipient. Formally, we start by setting $j_t^+(i) = j_t^-(i) = 0$ for all $i \in N$. Then every $k \in N$ for which $r_t^{-1}(k)$ is nonempty chooses ℓ u.a.r. from $r_t^{-1}(k)$ and sets $j_t^+(\ell) = j_t^-(k) = f(X'_t(k))$.
5. The work is shared and then each queue processes one job. Formally, for all i, $X_{t+1}(i)$ is set to $\max(0, X'_t(i) + j_t^+(i) - j_t^-(i) - 1)$.

FIG. 1. *The Markov chain X. The transition from X_t to X_{t+1}.*

3. Work-stealing functions. In this section, we state the properties that the work-stealing function, f, must satisfy for our analysis.

DEFINITION 1. *We assume that for a positive constant δ, the arrival rate λ is at most $1 - \delta$. Let c be a constant which is sufficiently large with respect to δ^{-1} (see the proof of Lemma 14) and let $\alpha = 4e(c+1)$. Let $z = \lceil \alpha \lg n \rceil$. Let $\nu = n^{-2} + n^{-\alpha}$ and let $T = \lceil \frac{2c}{1-\lambda/(1-\nu)} \lg n \rceil$. Note that T is positive as long as $\nu < \delta$, and we will consider values of n for which this is true. Let g be the function given by $g(\ell) = f(\ell - T)$. We will use the function g in our analysis of the work-stealing process. Suppose that a processor has ℓ units of work in its queue. If no units of work are generated or stolen during T steps, it will then have $\ell - T$ units. Finally, it may give away $f(\ell - T) = g(\ell)$ units to another processor which requests work. Let Φ_f be the smallest integer such that, for all $j \in \{0, \ldots, z\}$, $g^j(\Phi_f/n) \geq 2T$, where $g^j(y)$ denotes the j-fold application of function g to argument y. That is, $g^0(y) = 1$, $g^1(y) = g(y)$, $g^2(y) = g(g(y))$, and so on. If no such integer Φ_f exists, say $\Phi_f = \infty$. Informally, Φ_f is a quantity that is so large that if we start with Φ_f units of work and focus on the (at least Φ_f/n) units of work in some particular queue and allow this work to be stolen up to z times, the quantity of work remaining at every processor involved is at least $2T$. This idea will be made more precise later.*

We require the work-stealing function f to satisfy the following properties.

Property 1. $0 \le f(\ell) \le \ell/2$.

Property 2. $f(\ell)$ is monotonically nondecreasing in ℓ.

Property 3. Φ_f is finite.

Properties 1 and 2 are natural and easy to understand. We conclude this section by showing that many natural work-stealing functions which satisfy Properties 1 and 2 also satisfy Property 3. We start with a general lemma and then conclude with particular examples.

LEMMA 2. *Suppose that the work-stealing function f satisfies Properties* 1 *and* 2. *Let $h_0 = 2T$. Suppose that there are positive integers $h_1, \ldots, h_z$ satisfying $f(h_i - T) \ge h_{i-1}$. Then $\Phi_f \le nh_z$.*

Proof. We wish to show that for all $j \in \{0, \ldots, z\}$, $g^j(h_z) \ge 2T$. Since f satisfies Property 1, the condition $f(h_i - T) \ge h_{i-1}$ implies that $h_{i-1} \le h_i$. Therefore, for any $j \in \{0, \ldots, z\}$, $h_{z-j} \ge h_0 \ge 2T$. Thus, it suffices to prove $g^j(h_z) \ge h_{z-j}$, which we will do by induction on j with base case $j = 0$. For the inductive step, note that

$$g^{j+1}(h_z) = f(g^j(h_z) - T) \ge f(h_{z-j} - T) \ge h_{z-(j+1)},$$

where the first inequality uses the monotonicity of f (Property 2) and the inductive hypothesis. □

COROLLARY 3. *Suppose that the work-stealing function f satisfies Properties* 1 *and* 2. *Suppose that $f(\ell) = \omega(1)$ as a function of ℓ. Then f satisfies Property* 3.

Proof. Since $f(\ell) = \omega(1)$, the function gets arbitrarily big and the values $h_1, \ldots, h_z$ in Lemma 2 exist. □

Corollary 3 demonstrates that having $f(\ell) = \omega(1)$ is *sufficient* in the sense that this, together with Properties 1 and 2, implies Property 3. Having $f(\ell) = \omega(1)$ is not *necessary* though, as the following observation shows.

OBSERVATION 4. *Suppose that the work-stealing function f satisfies Properties* 1 *and* 2. *Let $h_0 = 2T$. Suppose (as in Lemma* 2*) that there are positive integers $h_1, \ldots, h_z$ satisfying $f(h_i - T) \ge h_{i-1}$. Let f' be the work-stealing function given by $f'(\ell) = \min(f(\ell), f(h_z))$. Then f' satisfies Properties* 1–3 *and has $\Phi_{f'} \le nh_z$.*

We end the section by giving an upper bound for Φ_f when f is a member of a popular class of work-stealing functions.

LEMMA 5. *Let $f(\ell) = \lfloor \ell/r \rfloor$ for some $r \ge 2$. This function satisfies Properties* 1–3 *and satisfies*

$$\Phi_f \le n(2T + 2r)(2r)^z.$$

Proof. We use Lemma 2. Let $h_i = (2T + 2r)(2r)^i$ for $i \in \{1, \ldots, z\}$. Then for $i \in \{1, \ldots, z\}$,

$$\begin{aligned} f(h_i - T) &= f((2T + 2r)(2r)^i - T) \\ &\ge \frac{(2T + 2r)(2r)^i}{r} - \frac{T}{r} - 1 \\ &= \frac{(2T + 2r)(2r)^i}{2r} + \frac{(2T + 2r)(2r)^i}{2r} - \frac{2T}{2r} - \frac{2r}{2r} \\ &\ge (2T + 2r)(2r)^{i-1} \\ &\ge h_{i-1}. \end{aligned}$$

□

Remark 6. The value Φ_f corresponding to the function f in Lemma 5 is bounded from above by a polynomial in n. To see this, note that the multiplier in the definition of T is

$$\frac{1}{1-\frac{\lambda}{1-\nu}} \le \frac{1}{1-\frac{1-\delta}{1-\nu}} = \frac{1-\nu}{\delta-\nu} \le \frac{1}{\delta-\nu} \le \frac{2}{\delta},$$

where the last inequality assumes $\nu \le \delta/2$, which is true if n is sufficiently large with respect to the constant δ^{-1}.

4. Upper bounds. In this section we prove that the system is stable for every work-stealing function satisfying Properties 1–3 in section 3. Our analysis does not depend upon the particular distribution $\mathcal{D}$ which governs the allocation of generators—the analysis works for an arbitrary distribution.

As already outlined in section 1, the basic idea is the following. The Markov chain X models our system. Since this chain is difficult to analyze directly, we introduce a second chain Z and investigate properties of Z instead. Then, using a coupling, we relate the results to chain X itself.

To put it *very* informally and nonrigorously, the core idea is to show that during an interval of length $T = O(\log n)$ not too many requests are sent. Since in our model not sending a request means servicing a task, we can show that in this case the system load decreases. Obviously, the crux is bounding the number of requests during the interval. Informally, this is done by assuming (for contradiction) that there are many requests during the interval, say at least R. Since the system load is initially high, there is at least one processor, processor P, which initially has a high load. This implies that after around $R' < R$ requests, we can view most of the requests that have been accepted in a tree with P at the root, and the leaves being processors that either directly or indirectly received a portion of P's initial load. By showing that (i) there are many leaves, and (ii) the tree does not grow very deep, we can conclude that after R' requests, there are many processors having a large load (at least T), and none of them will send a request during the next T steps. Hence, we can contradict the assumption that R requests get sent during the interval. Of course, this kind of proof-by-contradiction is invalid if we want to avoid conditioning the random variables during the T steps, so we have to do things more carefully.

4.1. Background. We start with some brief definitions regarding Markov chains. For more details, see [19]. The Markov chains that we consider are *time-homogeneous* (transition probabilities do not change over time) and *irreducible* (every state is reachable from every other) and *aperiodic* (the gcd of the lengths of valid paths from state i to itself is 1). An irreducible aperiodic Markov chain (Υ_t) is said to be *recurrent* if, with probability 1, it returns to its start state. That is, it is recurrent if

$$\Pr(\Upsilon_t = \Upsilon_0 \text{ for some } t \ge 1) = 1.$$

Otherwise, it is said to be *transient*. It is said to be *positive recurrent* or *ergodic* if the expected time that it takes to return to the start state is finite. In particular, let

$$T_{\text{ret}} = \min\{t \ge 1 \mid \Upsilon_t = \Upsilon_0\}.$$

The chain is said to be positive recurrent if $E[T_{\text{ret}}] < \infty$. A positive recurrent chain has a unique stationary distribution π. When we analyze the Markov chain X we will use the following generalization of Foster's theorem, due to Fayolle, Malyshev, and Menshikov (Theorem 2.2.4 of [16]).

LEMMA 7 (Foster; Fayolle, Malyshev, Menshikov [16]). *A time-homogeneous irreducible aperiodic Markov chain ζ with a countable state space Ω is positive recurrent if and only if there exists a positive function $\Phi(x)$, $x \in \Omega$, a number $\xi > 0$, a positive integer-valued function $\beta(x)$, $x \in \Omega$, and a finite set $C' \subseteq \Omega$ such that the following inequalities hold:*

$$E[\Phi(\zeta_{t+\beta(\zeta_t)}) - \Phi(\zeta_t) \mid \zeta_t = x] \le -\xi\beta(x), \qquad x \notin C', \tag{1}$$

$$E[\Phi(\zeta_{t+\beta(\zeta_t)}) \mid \zeta_t = x)] < \infty, \qquad x \in C'. \tag{2}$$

We also use the following Chernov–Hoeffding inequalities. The first of these is a special case of Theorem 4.2 of [31] and the second is taken from Theorem 5.7 of [24].

LEMMA 8 (Chernov). *Let $Z_1, \ldots, Z_s$ be independent Bernoulli trials with $\Pr(Z_i = 1) = p$. Let $\widehat{Z} = \sum_{i=1}^{s} Z_i$. Then for any ρ in $(0, 1]$, $\Pr(\widehat{Z} < (1-\rho)sp) \le \exp(-sp\rho^2/2))$.*

LEMMA 9 (Hoeffding). *Let $Z_1, \ldots, Z_s$ be independent random variables with $a_i \le Z_i \le b_i$ for suitable constants a_i, b_i and all $1 \le i \le s$. Also let $\widehat{Z} = \sum_{i=1}^{s} Z_i$. Then for any $t > 0$,*

$$\Pr\bigl(|\widehat{Z} - E(\widehat{Z})| \ge t\bigr) \le \exp\Bigl(-2t^2 \Big/ \sum_{i=1}^{s} (b_i - a_i)^2\Bigr).$$

4.2. Results. Our Markov chain X is time-homogeneous, irreducible, and aperiodic. Its state space is countable. Therefore, it satisfies the initial conditions of Lemma 7. We will prove the following theorem.

THEOREM 10. *Let δ be a positive constant and λ an arrival rate which is at most $1 - \delta$. Let f be a work-stealing function satisfying Properties 1–3 in section 3. Then for every n which is sufficiently large with respect to δ^{-1}, the Markov chain X is positive recurrent.*

Theorem 10 guarantees that the Markov chain X has a stationary distribution π. The next theorem is concerned with the value of the total system load in the stationary distribution. Recall from Definition 1 that $\nu = n^{-2} + n^{-\alpha}$. Our next theorem uses the following additional definitions.

DEFINITION 11. *Let ϵ be $(1 - \lambda/(1 - \nu))/4$. Let $\Phi(X_t)$ be the system load after step t. That is, $\Phi(X_t) = \sum_{i=1}^{n} X_t(i)$.*

THEOREM 12. *Let δ be a positive constant and λ an arrival rate which is at most $1 - \delta$. Let f be a work-stealing function satisfying Properties 1–3 in section 3. Then for every n which is sufficiently large with respect to δ^{-1},*

$$E_\pi[\Phi(X_t)] \le \Phi_f + 2nT/\epsilon + nT,$$

and for any nonnegative integer m,

$$\Pr_\pi[\Phi(X_t) > \Phi_f + 2nTm + nT] \;\le\; \exp(-\ln(1+\epsilon)(m+1)).$$

4.3. A simpler Markov chain. Let C be the set of states x with $\Phi(x) < \Phi_f$. In this section we define a simpler Markov chain Z that will be used in order to analyze the Markov chain X with a start state $x \notin C$.

The state space of Z is more complicated than the state space of X, but in some sense the information contained in a state of Z is less precise than the information contained in a state of X. In particular, a state Z_t consists of a tuple

$$(L_t(1), \ldots, L_t(n), Y_t(1), \ldots, Y_t(n)).$$

The variable $L_t(i)$ gives a crude indication of the load at processor i after step t. In any initial state that we will consider, *exactly one* processor (which we will call J_x) will have $L_0(J_x)$ large. All other processors will have $L_0(i) = 0$. Informally, these variables will have the following role for a processor i. Let t be the first time step during which processor i steals some of the work that originally sat at processor J_x. For $t' \leq t$, the variable $Y_{t'}(i)$ denotes the load of processor i and $L_{t'}(i) = 0$. For $t' > t$, the variable $L_{t'}(i)$ is positive and $Y_{t'}(i) = 0$. The exact value of $L_{t'}(i)$ gives an indication of how many times the work that processor i acquired at step t has been split (and, therefore, of how long it will last).

We will be observing the evolution of the Markov chain X starting at a state $X_0 = x$ with $\Phi(x) \geq \Phi_f$. This condition guarantees that for some $i \in N$, $X_0(i) \geq \Phi_f/n$. Let J_x be the smallest such i. In the following, we will be paying special attention to a load which originates at processor J_x. Thus, in the Markov chain Z, the state Z_0 which corresponds to X_0 is defined as follows. $L_0(J_x) = 2^z$ and $Y_0(J_x) = 0$. For all $i \neq J_x$, $L_0(i) = 0$ and $Y_0(i) = X_0(i)$. For convenience, we will say that a processor i is "heavily loaded" in state Z_t if and only if $L_t(i) > 0$. Thus, J_x is the only processor which is deemed to be "heavily loaded" in Z_0. Note that the state Z_0 is strictly a function of x. We will refer to this state as $Z(x)$. The transition from Z_t to Z_{t+1} is described in Figure 2. It may look surprising at first that the "heavy load" parameter $L_t(k)$ is halved every time a heavily loaded processor transfers load. This halving allows us to study the dissemination of load from J_x without considering the many dependent events.

Let R'_t be the set of requests made during the transition from Z_t to Z_{t+1}. (This transition is referred to as "step $t+1$.") That is, $R'_t = |\{i \mid r'_t(i) > 0\}|$. Let τ' be the smallest integer such that $R'_0 + \cdots + R'_{\tau'-1} \geq cn \lg n$. Let Ψ be the smallest integer such that, for some i, $L_\Psi(i) = 1$. Intuitively, $L_\Psi(i) = 1$ means that i has received load (directly or indirectly) from J_x (so it is "heavily loaded"), but this load has been

1. Choose the generator-allocation function h_t from $\mathcal{D}$.
2. If $L_t(i) > 0$, then $Y'_t(i)$ is defined to be 0 (just like $Y_t(i)$). Otherwise, $Y'_t(i)$ is defined to be $Y_t(i)$ plus the sum of $|h_t^{-1}(i)|$ Bernoulli random variables with mean λ.
3. $r'_t(i)$ is defined to be 0 *except* when $L_t(i) = 0$ and $Y'_t(i) = 0$. In this case, $r'_t(i)$ is chosen u.a.r. from N.
4. Start by setting
$$j_t^+(i) = j_t^-(i) = l_t^-(i) = l_t^+(i) = 0$$
for each $i \in N$. Then every $k \in N$ for which ${r'_t}^{-1}(k)$ is nonempty chooses ℓ u.a.r. from ${r'_t}^{-1}(k)$ and sets $l_t^+(\ell) = l_t^-(k) = L_t(k)/2$ and $j_t^+(\ell) = j_t^-(k) = f(Y'_t(k))$.
5. For all $i \in N$, $L_{t+1}(i)$ is set to $L_t(i) + l_t^+(i) - l_t^-(i)$. If $L_{t+1}(i) > 0$, then $Y_{t+1}(i) = 0$. Otherwise, $Y_{t+1}(i)$ is set to
$$\max(0, Y'_t(i) + j_t^+(i) - j_t^-(i) - 1).$$

FIG. 2. *The transition from Z_t to Z_{t+1}.*

split many times (it has been split z times, in fact). The following lemma shows that, with high probability, there are no such i and Ψ if at most $cn \log n$ requests are sent.

LEMMA 13. *Suppose $x \notin C$. Run Markov chain Z starting at $Z_0 = Z(x)$. Then*

$$\Pr(\Psi \leq \tau') \leq n^{-\alpha}.$$

Proof. Since at most n requests are sent in a single step, the total number of requests sent during steps $1, \ldots, \tau'$ is at most $(c+1)n \lg n$.

Recall the construction of $Z(x)$ from the beginning of section 4.3. In particular, there is one "heavily loaded" processor, J_x, with $L_0(J_x) = 2^z$. Every other processor i has $L_0(i) = 0$.

Imagine that the value $L_0(J_x) = 2^z$ corresponds to a collection of 2^z tokens which initially sit at processor J_x. The value $L_t(k)$ gives the number of tokens which sit at processor k following step t. This is always a power of 2. If $L_t(k) > 1$, then the instruction $l_t^+(\ell) = l_t^-(k) = L_t(k)/2$ in step 3 of the transition from Z_t to Z_{t+1} splits the collection of tokens sitting at processor k and transfers half of these tokens to processor ℓ. The event $\Psi \leq \tau'$ occurs if and only if some token has its group split z times during steps $1, \ldots, \tau'$.

What is the probability that a given token has its group split z times? This is at most

$$\binom{(c+1)n \lg n}{z} n^{-z} \leq \left(\frac{e(c+1) \lg n}{z} \right)^z.$$

The probability that there exists a token which has its group split z times is thus at most

$$\left(\frac{2e(c+1) \lg n}{z} \right)^z \leq \left(\frac{2e(c+1)}{\alpha} \right)^{\alpha \lg n} = n^{-\alpha}. \qquad \square$$

The next lemma shows that, with high probability, the number of requests sent during the observed T time steps is less than $cn \log n$. This means that we have very little idle time during this period, which in turn implies the decrease of the system load (as we will see later).

LEMMA 14. *Suppose $x \notin C$. Run Markov chain Z starting at $Z(x)$.*

$$\Pr(\tau' \leq T) \leq n^{-2}.$$

Proof. Recall that R'_t is the number of requests during the transition from Z_t to Z_{t+1}. In particular, $R'_t = |\{i \mid L_t(i) = 0 \wedge Y'_t(i) = 0\}|$. R'_t is a random variable which depends only upon the state Z_t and upon the host-distribution function h_t. In particular, every processor i with $L_t(i) = 0$ and $Y_t(i) = 0$ contributes 1 to R'_t independently with probability $(1-\lambda)^{|h_t^{-1}(i)|}$ and contributes 0 to R'_t otherwise.

To make the conditioning clear, we will let $R'(s,h)$ be the random variable whose distribution is the same as that of R'_t, conditioned on $Z_t = s$ and $h_t = h$.

By Lemma 9,

$$\Pr\left(|R'(s,h) - E(R'(s,h))| \geq \frac{cn \lg n}{8T} \right) \leq \exp\left(-2 \left(\frac{cn \lg n}{8T} \right)^2 / n \right).$$

Let σ denote $\exp(-2(\frac{cn \lg n}{8T})^2/n)$. Note that σ is exponentially small in n. (This follows from the definition of T.)

Say that (s, h) is "dangerous" if

$$E(R'(s,h)) \geq (cn \lg n)/(4T).$$

Note that if (Z_t, h_t) is not dangerous, then, with probability at least $1-\sigma$, the number of requests during the transition from Z_t to Z_{t+1} is at most

$$(cn \lg n)/(4T) + (cn \lg n)/(8T) \leq (cn \lg n)/(2T).$$

Now suppose that (s, h) is dangerous and let k be any processor. Then

$$\begin{aligned}\Pr({r'_t}^{-1}(k) = \emptyset \mid Z_t = s \wedge h_t = h) &\leq \sigma + \left(1 - \frac{1}{n}\right)^{\frac{cn \lg n}{8T}} \\ &\leq \sigma + \left(1 - \frac{1}{n}\right)^{\frac{\delta n}{18}} \\ &\leq 1 - \gamma\end{aligned}$$

for a small positive constant γ which depends upon δ (but not upon c or n). Let M_t denote the number of heavily loaded processors during step t, i.e.,

$$M_t = |\{i \mid L_t(i) > 0\}|.$$

Let ξ_t denote the number of heavily loaded processors during step t that don't get requested, i.e.,

$$\xi_t = |\{k \mid L_t(k) > 0 \wedge {r'_t}^{-1}(k) = \emptyset\}|.$$

If (s, h) is dangerous, then

$$E[\xi_t \mid Z_t = s \wedge h_t = h] \leq (1-\gamma)M_t.$$

Thus by Markov's inequality,

$$\Pr\big(\xi_t \geq (1-\gamma/2)M_t \mid Z_t = s \wedge h_t = h\big) \leq 1 - \frac{\gamma}{2-\gamma}.$$

If $\xi_t < (1-\gamma/2)M_t$, then at least $(\gamma/2)M_t$ of the M_t heavily loaded processors give away work, so $M_{t+1} \geq (1+\gamma/2)M_t$. We say that the step following on from a dangerous state is "useful" if this occurs. We have just seen that for every dangerous state (s, h), the probability that the next step is useful is at least $\frac{\gamma}{2-\gamma}$.

Thus, if we have D dangerous states during some time interval, the number of useful steps following them dominates (from above) the sum of D independent Bernoulli random variables with probability $p = \frac{\gamma}{2-\gamma}$. Applying Lemma 8 with $D = (2/p)\log_{1+\gamma/2}(n)$ and $\rho = 1/2$, we find that the probability that this sum is less than $\log_{1+\gamma/2}(n)$ is at most $\exp(-\log_{1+\gamma/2}(n)/4)$. This means that if we have D dangerous states, then the probability that there are at least $\log_{1+\gamma/2}(n)$ useful steps following them is at least $1 - \exp(-\log_{1+\gamma/2}(n)/4)$.

Now, if there are actually at least $\log_{1+\gamma/2}(n)$ useful steps during steps 1–t, $M_{t+1} = n$, so there can be no further dangerous states. We conclude that with probability at least $1 - \exp(-\log_{1+\gamma/2}(n)/4)$ there are at most D dangerous steps *ever*.

If we make c sufficiently large with respect to γ, then $D < c \lg n/2$.

Now we have that, except for probability $\exp(-\log_{1+\gamma/2}(n)/4)$, the dangerous steps contribute fewer than $cn \lg n/2$ requests (ever). Furthermore, except for probability at most σT, the nondangerous steps contribute at most $cn \lg n/2$ requests during the first T steps. Thus, the probability that $\tau' \leq T$ is at most

$$\exp(-\log_{1+\gamma/2}(n)/4) + \sigma T \leq n^{-2}. \qquad \square$$

Lemmas 13 and 14 imply the following.

COROLLARY 15. *Suppose $x \notin C$. Run Markov chain Z starting at $Z(x)$.*

$$\Pr(T < \tau' < \Psi) \geq 1 - n^{-\alpha} - n^{-2}.$$

LEMMA 16. *Suppose $x \notin C$. Run Markov chain Z starting at $Z(x)$. For any $t \leq \Psi$ and any $i \in N$, either $L_t(i) = 0$ or, for some $j \in \{0, \ldots, z\}$, $L_t(i) = 2^j$.*

Proof. The lemma is proved by induction on t with the base case $t = 0$. Consider the assignment

$$L_{t+1}(i) = (L_t(i) - l_t^-(i)) + l_t^+(i)$$

in the transition from Z_t to Z_{t+1} in Figure 1. If the second term in the expression, $l_t^+(i)$, is greater than zero, then it is equal to $L_t(k)/2$ for some k with $r'_t(i) = k$, so $L_t(i) = 0$. The first term in the expression, $L_t(i) - l_t^-(i)$, is either $L_t(i)$ or $L_t(i)/2$. Thus, either $L_{t+1}(i)$ is $L_t(i)$ or it is $L_t(k)/2$ for some k. Using the terminology from the proof of Lemma 13, $L_{t+1}(i) = 2^{z-m}$ means that the tokens that sit at processor i after step $t+1$ have had their group split m times. Since $t \leq \Psi$, $m \leq z$. $\square$

4.4. Proof of Theorem 10. Our first task is to relate the Markov chain X to the simpler Markov chain Z. Recall the definitions of τ', R'_t, and Ψ from section 4.3. Let R_t be the set of requests made during the transition from X_t to X_{t+1}. That is, $R_t = |\{i \mid r_t(i) > 0\}|$. Let τ be the smallest integer such that $R_0 + \cdots + R_{\tau-1} \geq cn \lg n$.

LEMMA 17. *If $x \notin C$, then*

$$\Pr(\tau \leq T \mid X_0 = x) \leq \nu.$$

Proof. A (Markovian) coupling[1] of the Markov chains X and Z is a stochastic process (X_t, Z_t) such that (X_t), considered marginally, is a faithful copy of X, and (Z_t), considered marginally, is a faithful copy of Z. We will describe a coupling starting from state $(x, Z(x))$. That is, in our coupling, $X_0 = x$ and $Z_0 = Z(x)$. The coupling will have the property that for all $t \leq \min(T, \tau', \Psi)$ and all i,

$$r'_t(i) = r_t(i). \tag{3}$$

From (3), we can conclude that whenever the Z chain satisfies $T < \tau' < \Psi$, the coupled X chain satisfies $T < \tau$. Thus,

$$\Pr(T < \tau \mid X_0 = x) \geq \Pr(T < \tau' < \Psi \mid Z_0 = Z(x)),$$

so the lemma follows from Corollary 15.

To give the details of the coupling, we will use the notation in Figures 1 and 2. Recall from Definition 1 that g is the function given by $g(y) = f(y - T)$, where f is

[1] The word "coupling" is normally used in reference to combining two copies of the same Markov chain, so we are using the word in a slightly nonstandard way.

the work-stealing function which is guaranteed to satisfy $g^j(\Phi_f/n) \geq 2T$ for a finite Φ_f and for all $j \in \{0, \ldots, z\}$.

Our coupling will satisfy the following invariants for any $i \in N$ and any $t \leq \min(T, \tau', \Psi)$:

(1) $r'_t(i) = r_t(i)$,
(2) $L_t(i) = 0$ implies $X_t(i) = Y_t(i)$, and
(3) $L_t(i) = 2^j$ implies $X_t(i) \geq g^{z-j}(\Phi_f/n) - t$.

As we observed above, our objective is to describe a coupling that satisfies invariant (1). The other invariants will help us to show that our constructed coupling is indeed a coupling in the sense that the marginal distributions are correct. The purpose of the third invariant is to ensure that, in the chain X, a node will not become empty soon if the corresponding node in chain Z is heavily loaded.

The coupling is as follows. We start with $X_0 = x$ and $Z_0 = Z(x)$. Recall the construction of $Z(x)$ from section 4.3. In particular, $L_0(J_x) = 2^z$ and $Y_0(J_x) = 0$. For every other i, $L_0(i) = 0$ and $Y_0(i) = x(i)$. Invariants (2) and (3) are satisfied for $t = 0$ since $X_0(J_x) \geq \Phi_f/n$.

Now the transition from (X_t, Z_t) to (X_{t+1}, Z_{t+1}) is given as follows. In part 1 of the transition, the same generator-allocation function h_t is chosen for both chains. The $X'_t(i)$ variables are defined in part 2 of the transition from X_t to X_{t+1}. In part 2 of the coupled transition from Z_t to Z_{t+1}, we set $Y'_t(i) = 0$ if $L_t(i) > 0$. Otherwise, we set $Y'_t(i) = X'_t(i)$. Note that, since invariant (2) held after step t, the $Y'_t(i)$ variables are set according to the correct marginal distribution. The $r_t(i)$ variables are defined in part 3 of the transition from X_t to X_{t+1}. In part 3 of the coupled transition from Z_t to Z_{t+1}, we set $r'_t(i) = r_t(i)$. To show that the marginal distribution is correct, we observe that if $L_t(i) = 0$, then we defined $Y'_t(i)$ to be $X'_t(i)$. Thus, the $r'_t(i)$ variables are assigned correctly. On the other hand, if $L_t(i) > 0$, then, by Lemma 16, $L_t(i) = 2^j$ for some $j \in \{0, \ldots, z\}$ so by invariant (3), $X_t(i) \geq g^{z-j}(\Phi_f/n) - t \geq 2T - t > 0$. Thus, $X'_t(i) > 0$, and $r'_t(i)$ is defined correctly. In part 4 of the transition from X_t to X_{t+1}, we do the following. For every $k \in N$ for which $r_t^{-1}(k)$ is nonempty, we choose ℓ u.a.r. from $r_t^{-1}(k)$. Since ${r'_t}^{-1}(k) = {r_t}^{-1}(k)$, we can make the same choice for k in part 4 of the transition from Z_t to Z_{t+1}.

We need to prove that the coupling maintains invariants (1), (2), and (3). Invariant (1) (the one that we actually want) is by construction. Invariant (2) is not too difficult. Lemma 16 shows that all of the variables $L_t(i)$ are nonnegative. Furthermore, the analysis in the proof of Lemma 16 reveals that $L_{t+1}(i) = 0$ implies $L_t(i) = 0$. (To see this, recall that $L_{t+1}(i) = (L_t(i) - l_t^-(i)) + l_t^+(i)$. The second of these terms is nonnegative, and the first is either $L_t(i)$ or $L_t(i)/2$.) Thus, whenever we have $L_{t+1}(i) = 0$ we have $Y_t(i) = X_t(i)$, and in the coupling we get $Y'_t(i) = X'_t(i)$. In part 5 of the transition from X_t to X_{t+1} we set $X_{t+1}(i) = \max(0, X'_t(i) + j_t^+(i) - j_t^-(i) - 1)$, and in part 5 of the transition from Z_t to Z_{t+1}, we set $Y_{t+1}(i) = \max(0, Y'_t(i) + j_t^+(i) - j_t^-(i) - 1)$. Thus we need only argue that the $j_t^+(i)$ and $j_t^-(i)$ variables get the same values in both copies. The $j_t^-(i)$ variable is the same since $Y'_t(i) = X'_t(i)$. The $j_t^+(i)$ variables are positive only if processor i made a request, namely, $r_t(i) = r'_t(i) = k$, for some k. Since $L_{t+1}(i) = 0$, we know $L_t(k) = 0$. Hence, $Y'_t(k) = X'_t(k)$, and the $j_t^+(i)$ values are indeed the same.

Finally, we need to prove that the coupling maintains invariant (3). Suppose that $L_{t+1}(i) = 2^j$. We wish to show that $X_{t+1}(i) \geq g^{z-j}(\Phi_f/n) - (t+1)$. First, suppose

$L_t(i) = 0$. In this case, $L_{t+1}(i) = L_t(k)/2$, where $r_t(i) = k$. Then

$$\begin{aligned} X_{t+1}(i) &\geq f(X_t(k)) - 1 \\ &\geq f(g^{z-j-1}(\Phi_f/n) - t) - 1 \\ &\geq f(g^{z-j-1}(\Phi_f/n) - T) - 1 \\ &= g^{z-j}(\Phi_f/n) - 1 \\ &\geq g^{z-j}(\Phi_f/n) - (t+1), \end{aligned}$$

where the first inequality follows from the transition in Figure 1 and the second inequality follows from the facts that invariant (3) held after step t and that f is monotonically nondecreasing (Property 2 in section 3). The third inequality also follows from the fact that f is monotonically nondecreasing.

Second, suppose $L_t(i) = 2^j$. In this case $r_t^{-1}(i)$ is empty, so

$$X_{t+1}(i) \geq X_t(i) - 1 \geq g^{z-j}(\Phi_f/n) - t - 1.$$

Finally, suppose $L_t(i) = 2^{j+1}$. (To see that these are the only cases, namely, that $L_t(i) \in \{0, 2^j, 2^{j+1}\}$, see the proof of Lemma 16.) In this case r_t^{-1} is nonempty, so

$$X_{t+1}(i) \geq X_t(i) - f(X_t(i)) - 1.$$

Since f satisfies $f(\ell) \leq \ell/2$ (Property 1 in section 3), we have

$$X_{t+1}(i) \geq f(X_t(i)) - 1,$$

which is the same as the first case. □

The next lemma shows that the load has an appropriate drift when $\tau > T$.

LEMMA 18. *If $x \notin C$, then*

$$E[\Phi(X_T) \mid (X_0 = x) \wedge (\tau > T)] \leq \Phi(x) - 2\epsilon nT.$$

Proof. Let A_t be the number of new jobs that arrive in the system during the transition from X_t to X_{t+1}. Namely,

$$A_t = \sum_{i \in N} (X'_t(i) - X_t(i)).$$

Let $Y = A_0 + \cdots + A_{T-1}$. Splitting $E[Y \mid X_0 = x]$ into two conditional expectations, conditioned on whether or not $\tau > T$, we find

$$\begin{aligned} &E[Y \mid (X_0 = x) \wedge (\tau > T)] \\ &= \frac{E[Y \mid X_0 = x] - \Pr(\tau \leq T \mid X_0 = x) E[Y \mid (X_0 = x) \wedge (\tau \leq T)]}{\Pr(\tau > T \mid X_0 = x)}. \end{aligned}$$

By Lemma 17, the denominator is at least $1 - \nu$. The numerator is at most $E[Y \mid X_0 = x]$, which is λnT, since during each of the T steps each of the n generators generates a new job independently with probability λ. Thus,

$$E[Y \mid (X_0 = x) \wedge (\tau > T)] \leq \frac{\lambda nT}{1 - \nu}.$$

If $\tau > T$, then the number of jobs serviced during steps 1–T is at least $nT - cn \lg n$. (If a processor does not make a request, then it certainly services a job.) Thus, the quantity

$$E[\Phi(X_T) \mid (X_0 = x) \wedge (\tau > T)]$$

is at most the initial load, $\Phi(x)$, plus the expected number of arrivals, which we have seen above is at most $\frac{\lambda nT}{1-\nu}$, minus the expected number of services, which is at least $nT - cn \lg n$. Putting all of this together, we get

$$\begin{aligned} E[\Phi(X_T) \mid (X_0 = x) \wedge (\tau > T)] &\leq \Phi(x) - \left(1 - \frac{\lambda}{1-\nu}\right) nT + cn \lg n \\ &\leq \Phi(x) - \frac{1 - \frac{\lambda}{1-\nu}}{2} nT \\ &= \Phi(x) - 2\epsilon nT, \end{aligned}$$

where the second inequality uses the definition of T in Definition 1 and the equality uses the definition of ϵ in Definition 11. ∎

LEMMA 19. *Suppose that n is sufficiently large with respect to δ^{-1}. If $x \notin C$, then*

$$E[\Phi(X_T) \mid X_0 = x] \leq \Phi(x) - \epsilon nT.$$

Proof.

$$\begin{aligned} E[\Phi(X_T) \mid X_0 = x] = &\Pr(\tau > T \mid X_0 = x) E[\Phi(X_T) \mid (X_0 = x) \wedge \tau > T] \\ &+ \Pr(\tau \leq T \mid X_0 = x) E[\Phi(X_T) \mid (X_0 = x) \wedge \tau \leq T]. \end{aligned}$$

By Lemma 18, this is at most

$$\Pr(\tau > T \mid X_0 = x)(\Phi(x) - 2\epsilon nT) + \Pr(\tau \leq T \mid X_0 = x) E[\Phi(X_T) \mid (X_0 = x) \wedge \tau \leq T].$$

Since at most n messages arrive per step, this is at most

$$\Pr(\tau > T \mid X_0 = x)(\Phi(x) - 2\epsilon nT) + \Pr(\tau \leq T \mid X_0 = x)(\Phi(x) + nT).$$

This can be rearranged as

$$\begin{aligned} &\Phi(x) - (1 - \Pr(\tau \leq T \mid X_0 = x))(2\epsilon nT) + \Pr(\tau \leq T \mid X_0 = x)(nT) \\ = \ &\Phi(x) - 2\epsilon nT + \Pr(\tau \leq T \mid X_0 = x)(2\epsilon nT + nT). \end{aligned}$$

By Lemma 17, this is at most

$$\Phi(x) - 2\epsilon nT + \nu(2\epsilon nT + nT) = \Phi(x) - \epsilon nT - (\epsilon nT - \nu 2\epsilon nT - \nu nT).$$

The lemma follows from the fact that

$$\nu \leq \frac{\epsilon}{2\epsilon + 1}, \tag{4}$$

which is true, provided that n is sufficiently large with respect to δ^{-1}. To establish (4), refer to Definitions 1 and 11. If n is sufficiently large, then $\nu \leq \delta/2$, so

$$4\epsilon = 1 - \frac{\lambda}{1-\nu} \geq 1 - \frac{1-\delta}{1-\nu} = \frac{\delta - \nu}{1-\nu} \geq \frac{\delta}{2}.$$

Also,

$$\nu \leq \frac{\delta/8}{2(\delta/8) + 1} \leq \frac{\epsilon}{2\epsilon + 1}. \qquad ∎$$

Combining Lemmas 19 and 7, we get a proof of Theorem 10.

4.5. Proof of Theorem 12. The proof of Theorem 12 uses the following theorem, which is Theorem 1 of [8].

LEMMA 20 (Bertsimas, Gamarnik, and Tsitsiklis [8]). *Consider a time-homogeneous Markov chain ζ with a countable state space Ω and stationary distribution π'. If there is a positive function $\Phi(x)$, $x \in \Omega$, a number $\xi > 0$, and a number $\beta \geq 0$ such that*

$$E[\Phi(\zeta_{t+1}) - \Phi(\zeta_t) \mid \zeta_t = x] \leq -\xi, \qquad \Phi(x) > \beta, \tag{5}$$

and

$$|\Phi(\zeta_{t+1}) - \Phi(\zeta_t)| \leq \nu_{\max}, \tag{6}$$

and, for any x,

$$\Pr[\Phi(\zeta_{t+1}) > \Phi(\zeta_t) \mid \zeta_t = x] \leq p_{\max} \tag{7}$$

and

$$E_{\pi'}[\Phi(\zeta_t)] < \infty, \tag{8}$$

then for any nonnegative integer m,

$$\Pr_{\pi'}[\Phi(\zeta_t) > \beta + 2\nu_{\max} m] \leq \left(\frac{p_{\max}\nu_{\max}}{p_{\max}\nu_{\max} + \xi}\right)^{m+1}$$

and

$$E_{\pi'}[\Phi(\zeta_t)] \leq \beta + \frac{2p_{\max}(\nu_{\max})^2}{\xi}.$$

Let $W_i = \Phi(X_{iT})$ for $i \in \{0, 1, 2, \dots\}$. Lemma 19 shows that the process $W_0, W_1, \dots$ behaves like a supermartingale above Φ_f. That is, it satisfies (5) with $\xi = \epsilon nT$ and $\beta = \Phi_f$. In itself, this does not imply that $E[W_t]$ is bounded (see Pemantle and Rosenthal's paper [32] for counterexamples). However, we also have

$$|W_{t+1} - W_t| \leq nT \tag{9}$$

for any t. That is, (6) is satisfied with $\nu_{\max} = nT$. This implies (for example, by Theorem 1 of [32] or by Theorem 2.3 of [20]) that $E_\pi[W_t]$ is finite (so (8) is satisfied). Lemma 20 can now be applied with $p_{\max} = 1$ to get

$$E_\pi[W_t] \leq \Phi_f + 2nT/\epsilon, \tag{10}$$

and for any nonnegative integer m,

$$\Pr_\pi[W_t > \Phi_f + 2nTm] \leq \left(\frac{nT}{nT + \epsilon nT}\right)^{(m+1)}. \tag{11}$$

The theorem now follows from the observation that for $0 \leq j < T$,

$$\Phi(X_{iT+j}) \leq \Phi(X_{iT}) + nj.$$

5. Lower bounds. In this section we give the straightforward lower bound, which shows that the system is not stable for unsuitable work-stealing functions. Of course we have to put some restrictions on $\mathcal{D}$ in order to obtain instability. For example, if $\mathcal{D}$ is the point distribution containing a single function h which allocates one generator to each processor, then the system will be stable even without any work-stealing.

The proof of our lower bound uses the following lemma, which is Theorem 2.2.7 of [16].

LEMMA 21 (Fayolle, Malyshev, and Menshikov [16]). *An irreducible aperiodic time-homogeneous Markov chain ζ with countable state space Ω is transient if there is a positive function Φ with domain Ω and there are positive constants C, d, and ξ such that*

1. *there is a state x with $\Phi(x) > C$, and a state x with $\Phi(x) \le C$,*
2. *$E[\Phi(\zeta_1) - \Phi(\zeta_0) \mid \zeta_0 = x] \ge \xi$ for all x with $\Phi(x) > C$, and*
3. *if $|\Phi(x) - \Phi(y)| > d$, then the probability of moving from x to y in a single move is 0.*

If we use $k = 1$ in the statement of the following theorem, we find that the system is unstable if $f(\ell) < \lambda n - 1$.

THEOREM 22. *Let δ be a positive constant and λ an arrival rate which is at most $1 - \delta$. Suppose that $\mathcal{D}$ contains a single generator-allocation function h which distributes the n generators equally among some set of k processors. Suppose that for all ℓ, $f(\ell) \le j(n)$. Then the Markov chain X is transient if*

$$k \cdot (j(n) + 1) < \lambda n.$$

Proof. This theorem can be proven easily using Lemma 21. Recall that the start state X_0 is $(0, \ldots, 0)$ (all queues are initially empty). First, we bound the amount of work that can be done during any given step. When a processor steals work, it only gets enough work for at most $j(n)$ rounds. Since each processor gives work to only one other processor per round, and there are at most k processors with generators, at most $j(n)k$ processors without generators have work to do during any given step. Thus, at most $(j(n) + 1)k$ tasks can be done during any step. The expected load increase of the system during a step is λn. Using Lemma 21 with Φ as the system load, it is easy to see that the system is transient if $k(j(n) + 1) < \lambda n$. □

6. Conclusions. We have analyzed a very simple work-stealing algorithm, which is successfully being used in practical applications. In this paper we have analyzed its performance for a wide range of parameters. We have shown that it is stable for any constant generation rate $\lambda < 1$ and a wide class of work-stealing functions f. On the other hand, we have shown that for every $\lambda > 0$ there is a class of unsuitable work-stealing functions, for which it is not stable. Finally, we have derived upper bounds on the system load when the system is stable.

It would be interesting to know whether there is a nice characterization of the class of functions that lead to stability. It would also be interesting to know how far our upper bounds on system load are from the truth. We suspect that the system load is actually much smaller than our upper bounds indicate, but it would be useful to have rigorous experimental results.

Acknowledgments. We thank Hesham Al-Ammal for useful discussions. We also thank the referees, both of whom provided many helpful comments.

REFERENCES

[1] M. Adler, P. Berenbrink, and K. Schröder, *Analyzing an infinite parallel job allocation process*, in Proceedings of the 6th European Symposium on Algorithms (ESA'98), Lecture Notes in Comput. Sci., Springer-Verlag, New York, 1998, pp. 417–428.

[2] M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen, *Parallel randomized load balancing*, in Proceedings of the 27th Symposium on Theory of Computing (STOC'95), ACM Press, New York, 1995, pp. 234–247.

[3] H. Al-Ammal, L. A. Goldberg, and P. MacKenzie, *An improved stability bound for binary exponential backoff*, Theory Comput. Systems, 34, (2001), pp. 229–244.

[4] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, *Balanced allocations (extended abstract)*, in Proceedings of the 26th Symposium on Theory of Computing (STOC'94), ACM Press, New York, 1994, pp. 593–602.

[5] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking, *Balanced allocations: The heavily loaded case*, in Proceedings of the 32th ACM Symposium on Theory of Computing (STOC'00), ACM Press, New York, 2000, pp. 745–754.

[6] P. Berenbrink, T. Friedetzky, and E. W. Mayr, *Parallel continuous randomized load balancing*, in Proceedings of the 10th Symposium on Parallel Algorithms and Architectures (SPAA '98), ACM Press, New York, 1998, pp. 192–201.

[7] P. Berenbrink, T. Friedetzky, and A. Steger, *Randomized and adversarial load balancing*, in Proceedings of the 11th Symposium on Parallel Algorithms and Architectures (SPAA'99), ACM Press, New York, 1999, pp. 175–184.

[8] D. Bertsimas, D. Gamarnik, and J. N. Tsitsiklis, *Performance of multiclass Markovian queueing networks via piecewise linear Lyapunov functions*, Ann. Appl. Probab., 11 (2001), pp. 1384–1428.

[9] R. Blumofe and C. Leiserson, *Scheduling multithreaded computations by work stealing*, in Proceedings of the 35th Symposium on Foundations of Computer Science (FOCS'94), IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 356–368.

[10] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, *Cilk: An efficient multithreaded runtime system*, J. Parallel Distrib. Comput., 37 (1996), pp. 55–69.

[11] A. Borovkov, *Ergodicity and Stability of Stochastic Processes*, John Wiley and Sons, New York, 1998.

[12] R. Cole, A. Frieze, B. Maggs, M. Mitzenmacher, A. Richa, R. Sitaraman, and E. Upfal, *On balls and bins with deletions*, in Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '98), Lecture Notes in Comput. Sci., Springer-Verlag, New York, 1998, pp. 145–158.

[13] A. Czumaj and V. Stemann, *Randomized allocation processes*, in Proceedings of the 38th Symposium on Foundations on Computer Science (FOCS'97), IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 194–203.

[14] T. Decker, *Virtual Data Space—Load balancing for irregular applications*, Parallel Comput., 26 (2000), pp. 1825–1860.

[15] P. Fatourou and P. Spirakis, *Scheduling algorithms for strict multithreaded computations*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC'96), Lecture Notes in Comput. Sci., Springer-Verlag, New York, 1996, pp. 407–416.

[16] G. Fayolle, V. Malyshev, and M. Menshikov, *Topics in the Constructive Theory of Countable Markov Chains*, Cambridge University Press, Cambridge, UK, 1995.

[17] B. M. Feldmann, P. Mysliwietz, and B. Monien, *Studying overheads in massively parallel min/max-tree evaluation*, in Proceedings of the 5th Symposium on Parallel Algorithms and Architectures, ACM Press, New York, 1994, pp. 94–103.

[18] L. Goldberg and P. MacKenzie, *Analysis of practical backoff protocols for contention resolution with multiple servers*, J. Comput. Systems Sci., 58 (1999), pp. 232–258.

[19] G. Grimmet and D. Stirzaker, *Probability and Random Processes*, 2nd ed., Oxford University Press, Oxford, 1992.

[20] B. Hajek, *Hitting time and occupation time bounds implied by drift analysis with applications*, Adv. Appl. Probab., 14 (1982), pp. 502–525.

[21] J. Håstad, T. Leighton, and B. Rogoff, *Analysis of backoff protocols for multiple access channels*, SIAM J. Comput., 25 (1996), pp. 740–774.

[22] R. Lüling and B. Monien, *A dynamic distributed load balancing algorithm with provable good performance*, in Proceedings of the 5th Symposium on Parallel Algorithms and Architectures (SPAA'93), ACM Press, New York, 1993, pp. 164–172.

[23] N. R. Mahapatra and S. Dutt, *Adaptive quality equalizing: High-performance load balancing*

for parallel branch and bound across applications and computing systems, in Proceedings of the Joint IEEE Parallel Processing Symposium/Symposium on Parallel and Distributes Processing, IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 796–800.

[24] C. McDiarmid, *On the method of bounded differences*, in Surveys in Combinatorics, London Math. Soc. Lecture Note Ser. 141, Cambridge University Press, Cambridge, UK, 1989, pp. 148–188.

[25] S. Meyn and R. Tweedie, *Markov Chains and Stochastic Stability*, Springer-Verlag, London, 1993.

[26] M. Mitzenmacher, *Density dependent jump Markov processes and applications to load balancing*, in Proceedings of the 37th Symposium on Foundations of Computer Science (FOCS'96), IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 213–222.

[27] M. Mitzenmacher, *The Power of Two Random Choices in Randomized Load Balancing*, Ph.D. thesis, Graduate Division, University of California at Berkeley, 1996.

[28] M. Mitzenmacher, *On the analysis of randomized load balancing schemes*, in Proceedings of the 9th Symposium on Parallel Algorithms and Architectures (SPAA'97), ACM Press, New York, 1997, pp. 292–301.

[29] M. Mitzenmacher, *Analysis of load stealing models based on differential equations*, in Proceedings of the 10th Symposium on Parallel Algorithms and Architectures (SPAA'98), ACM Press, New York, 1998, pp. 212–221.

[30] M. Mitzenmacher, A. Richa, and R. Sitaraman, *The power of two randomized choices: A survey of techniques and results*, in Handbook of Randomized Computing, Kluwer Academic, Dordrecht, The Netherlands, 2001, pp. 255–305.

[31] R. Motwani and P. Raghavan, *Randomized Algorithms,* Cambridge University Press, Cambridge, UK, 1995.

[32] R. Pemantle and J. S. Rosenthal, *Moment conditions for a sequence with negative drift to be uniformly bounded in l^r*, Stochastic Process. Appl., 82 (1999), pp. 143–155.

[33] L. Rudolph, M. Slivkin-Allalouf, and E. Upfal, *A simple load balancing scheme for task allocation in parallel machines*, in Proceedings of the 3rd Symposium on Parallel Algorithms and Architectures (SPAA '91), ACM Press, New York, 1991, pp. 237–245.

[34] B. Vöcking, *How asymmetry helps load balancing*, in Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS'00), IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 131–140.

[35] N. D. Vvedenskaya, R. L. Dobrushin, and F. I. Karpelevich, *Queueing system with selection of the shortest of two queues: An asymptotic approach*, Problems Inform. Transmission, 32 (1996), pp. 15–27.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1280–1316

THE ALGEBRAIC APPROACH TO THE DISCRETE COSINE AND SINE TRANSFORMS AND THEIR FAST ALGORITHMS*

MARKUS PÜSCHEL† AND JOSÉ M. F. MOURA†

Abstract. It is known that the discrete Fourier transform (DFT) used in digital signal processing can be characterized in the framework of the representation theory of algebras, namely, as the decomposition matrix for the regular module $\mathbb{C}[Z_n] = \mathbb{C}[x]/(x^n - 1)$. This characterization provides deep insight into the DFT and can be used to derive and understand the structure of its fast algorithms. In this paper we present an algebraic characterization of the important class of discrete cosine and sine transforms as decomposition matrices of certain regular modules associated with four series of Chebyshev polynomials. Then we derive most of their known algorithms by pure algebraic means. We identify the mathematical principle behind each algorithm and give insight into its structure. Our results show that the connection between algebra and digital signal processing is stronger than previously understood.

Key words. discrete cosine transform (DCT), discrete sine transform (DST), discrete trigonometric transform (DTT), discrete Fourier transform (DFT), FFT, polynomial transform, fast algorithm, Chebyshev polynomial, algebra representation, group representation, symmetry

AMS subject classifications. Primary, 42C05, 42C10, 33C80, 33C90, 65T50, 65T99; Secondary, 15A23, 62-07

DOI. 10.1137/S009753970139272X

1. Introduction.

Many algorithms in digital signal processing are based on the use of linear discrete signal transforms. Mathematically, such a transform is a matrix-vector multiplication $a \mapsto M \cdot a$, where $a \in \mathbb{F}^n$ is the sampled signal, and $M \in \mathbb{F}^{n \times n}$ is the transform over some base field $\mathbb{F}$. We will consider only $\mathbb{F} = \mathbb{C}$. Crucial for the applicability of a signal transform M is the existence of fast algorithms that allow its computation within $O(n \log n)$ operations (or better), compared with $O(n^2)$ arising from a direct matrix-vector multiplication. The problem of finding these algorithms for different transforms has been a major research topic leading to a vast number of publications in signal processing and mathematics.

In this paper we present an algebraic approach to the class of the 16 trigonometric transforms in the framework of algebra representation theory. Then we use algebraic methods to derive most of their known fast algorithms. Our results give insight into the structure and the existence of these algorithms and extend the relationship between signal processing and algebra that is currently restricted mainly to the discrete Fourier transform (DFT).

1.1. Transforms and algorithms.

Probably the most famous example of a signal transform is the DFT, which is used in harmonic analysis to decompose a signal into its frequency components. Important algorithms for the DFT include the "fast Fourier transform" (FFT) found by Cooley and Tukey [11] (originally due to Gauß; see [23]), Rader's algorithm for prime size [39], Winograd's algorithms [54], and several others. An overview on DFT algorithms can be found, for example, in [49].

Another important class of transforms consists of the eight different types of dis-

*Received by the editors July 19, 2001; accepted for publication (in revised form) May 23, 2003; published electronically August 15, 2003. This work was supported by the NSF through award 9988296.

http://www.siam.org/journals/sicomp/32-5/39272.html

†Department of Electrical and Computer Engineering, Carnegie Mellon University, Forbes Ave. 5000, Pittsburgh, PA 15213 (pueschel@ece.cmu.edu, moura@ece.cmu.edu).

crete cosine and sine transforms (DCTs and DSTs, respectively), also called discrete trigonometric transforms (DTTs). Their applications include image and video compression [40]. Important algorithms for the trigonometric transforms were developed by Chen, Smith, and Fralick [7], Wang [52], Yip and Rao [56, 57], Vetterli and Nussbaumer [50], Lee [28], Feig [20], Chan and Ho [6], Steidl and Tasche [46], and Feig and Winograd [21].

Each of these algorithms has been derived through insightful manipulation of the transform matrix entries. The algorithms are highly structured, a property that can be used to write them as sparse factorizations of the transform matrix in a very concise way using mathematical operators. As examples, the Cooley–Tukey FFT can be written as

$$\mathrm{DFT}_{mn} = (\mathrm{DFT}_m \otimes \mathrm{I}_m) \cdot D \cdot (\mathrm{I}_m \otimes \mathrm{DFT}_n) \cdot P, \tag{1.1}$$

and an example of an algorithm for the DCT-2 is

$$\text{DCT-2}_{2n} = Q \cdot (\text{DCT-2}_n \oplus \text{DCT-4}_n) \cdot B. \tag{1.2}$$

The notation will be explained in section 2; the matrices D, P, Q, B are all sparse. Both algorithms are of a recursive nature.

1.2. The algebraic characterization of the DFT. It is well known that a DFT (of size n) can be introduced in strict algebraic terms as the decomposition matrix for the group algebra $\mathbb{C}[Z_n]$ of a cyclic group Z_n with n elements,

$$\mathrm{DFT}_n : \ \mathbb{C}[Z_n] \to \mathbb{C} \oplus \cdots \oplus \mathbb{C}, \tag{1.3}$$

or, equivalently, as the decomposition matrix for the algebra $\mathbb{C}[x]/(x^n - 1)$,

$$\mathrm{DFT}_n : \ \mathbb{C}[x]/(x^n - 1) \to \mathbb{C}[x]/(x - \omega_n^0) \oplus \cdots \oplus \mathbb{C}[x]/(x - \omega_n^{n-1}), \tag{1.4}$$

with respect to appropriate bases in each case. These decompositions are instantiations of the Wedderburn theorem for the semisimple algebra $\mathbb{C}[Z_n] \cong \mathbb{C}[x]/(x^n - 1)$. Equations (1.3) and (1.4) show that the DFT is indeed an algebraic object and thus provides a deep understanding of its use in signal processing. Furthermore, (1.3) and (1.4) can be used to easily derive and explain the structure of fast DFT algorithms by algebraic constructions rather than by manipulation of the DFT entries. As an example, (1.1) arises from a stepwise decomposition of $\mathbb{C}[Z_n]$, as has been shown by Auslander, Feig, and Winograd [2] and Beth [3].

Given the algebraic characterization of the DFT, we naturally obtain the following question: Is it possible to generalize (1.3) or (1.4) to capture a larger class of signal transforms in an algebraic framework? And, in the affirmative case: How do we use the algebraic characterization to derive and explain their fast algorithms?

1.3. Beyond the DFT. Depending on the interpretation of the DFT in (1.3) and (1.4), there have been two threads of generalization.

First, (1.3) has been generalized to arbitrary finite groups $G \neq Z_n$, leading to the area of "Fourier analysis on groups" that provides a rich class of transforms and the theory to derive their fast algorithms. Examples of important results in this field include the work of Beth [4], Clausen [9], Diaconis and Rockmore [13], and Rockmore [43]; a nice overview on this area can be found in [10] and in the survey articles [29] and [44]. However, with few exceptions, the Fourier transforms on groups do

not correspond to the transforms actually used in signal processing. This problem initiated a further generalization by Minkwitz [32, 31] to include $\mathbb{C}[G]$-modules that afford an arbitrary permutation representation. A matrix that decomposes such a module was said to have "symmetry." Minkwitz discovered that the DCT (type 3) possesses such a symmetry and derived, by pure algebraic means, a fast algorithm. That approach was further generalized by Egner and Püschel to include monomial representations: A decomposition theory [37, 36] and tools to analyze a matrix for symmetry and automatically derive a factorization [19, 15, 17] were developed. In [16] these tools were successfully applied to several signal transforms. Among the DTTs, the DCT and DST of type 3 and 4 exhibited symmetry and could be decomposed by these techniques.

Second, the generalization of (1.4) to arbitrary polynomials $p(x) \neq x^n - 1$ and arbitrary bases of $\mathbb{C}[x]/p(x)$ leads to the class of "polynomial transforms." If p is arbitrary and $(1, x, \ldots, x^{n-1})$ is chosen as a basis, one obtains a Vandermonde matrix, which is known to have a sparse factorization (see e.g., [5]). Driscoll, Healy, and Rockmore [14] developed a fast algorithm for the case of arbitrary (separable) polynomials p and bases consisting of orthogonal polynomial sequences. Independently, Potts, Steidl, and Tasche provided a numerically stable version of this algorithm [35]. In their paper the DCT of type 1 is recognized as a polynomial transform. Steidl and Tasche [46] also recognized the DCT of type 3 as a polynomial transform and used this property to derive a fast algorithm. In a different context, the DCTs and DSTs of types 1–4 have been related to polynomial transforms, in some cases after appropriate normalization [26].

Taken together, we encounter the following situation with respect to the DTTs:

1. There are 16 types of DTTs and a large number of publications on their fast algorithms.
2. In signal processing the DTTs are characterized as eigenmatrices of certain linear time-invariant processes with given boundary conditions [33, 47].
3. Four DTTs have been shown to exhibit group symmetries, and, in each case, an algorithm has been derived by algebraic means.
4. Two DTTs have been shown to be polynomial transforms. (Note that this property is not equivalent to point 3.) In one case this has been used to derive a fast algorithm. Further, six DTTs have been recognized as polynomial transforms after suitable normalization.

This sets the framework for the results presented in this paper.

1.4. The algebraic characterization of the DTTs. In this paper we present the algebraic characterization of the DTTs. This shows that, like the DFT, the DTTs are algebraic objects. Then we use the algebraic framework to derive, and explain, most of the fast DTT algorithms known in the literature. The results extend our previous preliminary work [38].

In particular we present the following:

1. A complete algebraic characterization of all 16 DTTs as scaled polynomial transforms (a generalization of polynomial transforms to be defined) arising from polynomial algebras $A = \mathbb{C}[x]/p(x)$ and A-modules of the form $f \cdot A$, where f is a scaling function. The construction of these modules follows the defining signal processing properties of the DTTs as eigenmatrices of certain linear time-invariant processes with given boundary conditions. Thus, our construction relates the domain of signal processing to the domain of algebra representation theory. As polynomials, four series of Chebyshev polynomials will naturally come into play.

2. A comprehensive overview of existing fast algorithms and their derivation by pure algebraic means, i.e., by manipulating modules and algebras rather than matrix entries. The algorithms are divided into classes depending on the mathematical principle that accounts for their existence. Examples, based on a direct manipulation of the A-module M and polynomial $p(x)$ associated with a DTT, include: (a) translation of a DTT into another DTT by a sparse base change in M; (b) recursive decomposition based on a factorization of p; and (c) recursive decomposition based on a decomposition of p. We continue our investigation by deriving a striking property of the DTTs. The characterization of the DTTs as scaled polynomial transforms, i.e., in a framework of polynomial algebras and their representations, leads naturally to group symmetry properties, i.e., properties in the framework of groups and their representations. We will identify two ways in which group symmetries come into play (a) by extending the A-module M to a suitable $\mathbb{C}[G]$-module, where G is a finite group, and (b) through certain subgroups of the automorphism group of A. These symmetry properties lead to algorithms that are structurally different from the ones obtained by direct derivation (see above). All techniques used for the derivation of fast DTT algorithms are potentially more generally applicable.

Taken together, our results provide a comprehensive framework that puts previous results on the DTTs into a common context, thus tying together their signal processing properties, their algebraic properties, and the structure of their fast algorithms.

1.5. Organization. The paper is divided into two parts. Part I (sections 2–6) provides the mathematical framework and establishes the algebraic interpretation of the DTTs. Part II (sections 7–10) uses different algebraic methods to derive and explain most of the known fast algorithms for the DTTs.

Part I. In section 2 we briefly describe the notation and mathematical concepts we use. Polynomial transforms and scaled polynomial transforms are introduced in section 3, together with their module-theoretic interpretations. In section 4 we present a generalization of Chebyshev polynomials, with particular attention to four special series and their properties. The 16 types of DTTs are introduced in section 5, together with their defining signal processing properties. In section 6 we construct for each DTT, using its signal processing properties, an associated module, which reveals that the DTTs are scaled polynomial transforms.

Part II. In section 7 we present general methods for obtaining fast algorithms for polynomial transforms and discuss results known from the literature. In section 8 we use the algebraic properties of the DTTs to derive and understand several known fast algorithms for the DTTs. Other classes of algorithms for the DTTs are explained in section 9 and are based on group representation symmetries. In section 10 we will briefly discuss algorithms that are not covered by the previous methods.

2. Notation and terminology. We will use the following notation and mathematical background.

Matrices. An $(n \times n)$-matrix with entry $a_{k,\ell}$ at row k and column ℓ is written as $[a_{k,\ell}]$. In most cases we provide the index range of k, ℓ as subscript. We denote by $A \oplus B = \left[\begin{smallmatrix} A & \\ & B \end{smallmatrix}\right]$ the direct sum of A and B. If $A = [a_{k,\ell}]$, then $A \otimes B = [a_{k,\ell} \cdot B]$ denotes the tensor or Kronecker product of A and B. The conjugation is written as $A^B = B^{-1} \cdot A \cdot B$. A monomial matrix has exactly one nonzero entry in every row and column. If σ is a permutation (usually written in cycle notation), we will denote a corresponding $(n \times n)$-matrix as $[\sigma, n]$, which has ones at positions $(i, \sigma(i))$. The special case $\sigma : i \mapsto ki \bmod n - 1$, $i = 0, \ldots, n-2$, and $n-1 \mapsto n-1$, for $k \mid n$, is

called "stride permutation," and we write $[\sigma, n] = \mathrm{L}^n_k$. A diagonal matrix is written as $\mathrm{diag}(L)$, where L is the list of diagonal elements. A monomial matrix is denoted by $[\sigma, L] = [\sigma, |L|] \cdot \mathrm{diag}(L)$.

Polynomials. Polynomials are denoted by lowercase letters, $p(x)$, $q(x)$, etc. We will often drop the argument for convenience. A polynomial is called separable if its zeros are pairwise distinct; i.e., if $\deg(p) = n$, then

$$p(x) = \prod_{k=0}^{n-1} (x - \alpha_k), \quad \alpha_i \neq \alpha_j, \quad \text{for } i \neq j.$$

Algorithms. If B is an $(n \times n)$-matrix, we mean by "a fast algorithm for B" a fast algorithm for computing the matrix vector product $x \mapsto B \cdot x$. Algorithms are given by factorizations, $B = B_1 \cdots B_k$, where all B_i are sparse. If we refer to the arithmetic cost of an algorithm or the arithmetic complexity of matrices B, we mean the number of additions and multiplications different from $1, -1$ (cf. [5]).

Algebras and modules. We assume that the reader is familiar with the basic theory of algebras and modules. Examples of introductory books on this topic are [12, 25]. All algebras in this paper are $\mathbb{C}$-algebras. In particular, we will consider the polynomial algebra $\mathbb{C}[x]$ and factor algebras $\mathbb{C}[x]/p(x)$, where p is a separable polynomial, and group algebras $\mathbb{C}[G]$, where G is a finite group. Each of the algebras $A = \mathbb{C}[x]/p(x)$ or $A = \mathbb{C}[G]$ is of finite dimension and semisimple; i.e., every finite-dimensional (left or right) A-module can be decomposed into a direct sum of irreducible submodules,

$$M \cong M_1 \oplus \cdots \oplus M_k,$$

which is called the *Wedderburn decomposition* of M. If bases in M and M_i, $i = 1, \ldots, k$, are chosen, then this isomorphism can be expressed by a matrix, which we will refer to as a *Wedderburn matrix*. The module M is usually a left module unless otherwise stated. If M has dimension n as $\mathbb{C}$ vector space, and a basis is chosen, then M affords a matrix representation of A, i.e., a homomorphism

$$\phi : \ A \to \mathbb{C}^{n \times n}.$$

The Wedderburn decomposition of M is equivalent to a decomposition of ϕ into a direct sum of irreducible representations. In the special case where $M \cong A$ (as A-modules), M is called the *regular* A-module, and a corresponding representation is also called regular.

The *annihilator* of M in A is defined as

$$\mathrm{ann}_A(M) = \{a \in A \mid a \cdot m = 0, \text{for all } m \in M\};$$

it is a two-sided ideal in A. If M is an A-module, then M is also an $A/\mathrm{ann}_A(M)$-module.

3. Polynomial algebras, modules, and transforms. In this section we introduce polynomial transforms and their algebraic interpretation as decomposition matrices of polynomial algebras. Then we extend the definition to the more general class of "scaled" polynomial transforms. This extension will enable us to capture all trigonometric transforms in an algebraic framework.

3.1. Polynomial transforms. Let

$$p(x) = \prod_{k=0}^{n-1} (x - \alpha_k)$$

be a separable polynomial. Then the algebra $A = \mathbb{C}[x]/p(x)$ is semisimple, and the Wedderburn decomposition of (the regular module) $M = A$ is given by the Chinese remainder theorem (CRT) as

$$\mathbb{C}[x]/p(x) \cong \bigoplus_{k=0}^{n-1} \mathbb{C}[x]/(x - \alpha_k). \tag{3.1}$$

We want to represent the isomorphism in (3.1) by a matrix.

DEFINITION 3.1 (polynomial transform). *Let $b = (p_0, \ldots, p_{n-1})$ be a basis of polynomials in $\mathbb{C}[x]/p(x)$, p separable, and $\alpha = (\alpha_0, \ldots, \alpha_{n-1})$ the vector of zeros of p, and assume that the one-dimensional algebras $\mathbb{C}[x]/(x - \alpha_k)$ have the base vector $1 = x^0$, respectively. With these choices, the isomorphism* (3.1) *is given by the $(n \times n)$-matrix*

$$\mathcal{P}_{b,\alpha} = [p_\ell(\alpha_k)]_{k,\ell=0,\ldots,n-1}, \tag{3.2}$$

where k is the row index. The Wedderburn matrix $\mathcal{P}_{b,\alpha}$ is called the polynomial transform *w.r.t. the polynomials b and the sample points α. (Note that the order of base polynomials and sample points matters.)*

The polynomial transform $\mathcal{P}_{b,\alpha}$ can also be characterized via the representation ϕ afforded by the module $A = M$ with basis b. This is the subject of the following lemma.

LEMMA 3.2. *We use previous notation. Let $p(x) = \prod_{k=0}^{n-1}(x - \alpha_k)$ be separable, $A = \mathbb{C}[x]/p(x)$, and $M = A$ be the (regular) left module with basis $b = (p_0, p_1, \ldots, p_{n-1})$ and polynomial transform $\mathcal{P}_{b,a}$. Let ϕ be the corresponding representation of A. Then*

(i) *$\mathcal{P}_{b,\alpha}^{-1}$ decomposes ϕ into a direct sum of irreducible representations. More precisely,*

$$\mathcal{P}_{b,\alpha} \cdot \phi(q(x)) \cdot \mathcal{P}_{b,\alpha}^{-1} = \operatorname{diag}(q(\alpha_0), \ldots, q(\alpha_{n-1})) \quad \text{for } q(x) \in A.$$

All such decomposition matrices are given by $\mathcal{P}_{b,\alpha}^{-1} \cdot D$, where D is diagonal and invertible.

(ii) *$\mathcal{P}_{b,\alpha}^{T}$ decomposes ϕ^T into a direct sum of irreducible representations. More precisely,*

$$(\mathcal{P}_{b,\alpha}^{T})^{-1} \cdot \phi^T(q(x)) \cdot \mathcal{P}_{b,\alpha}^{T} = \operatorname{diag}(q(\alpha_0), \ldots, q(\alpha_{n-1})) \quad \text{for } q(x) \in A.$$

All such decomposition matrices are given by $\mathcal{P}_{b,\alpha}^{T} \cdot D$, where D is diagonal and invertible.

Proof. The matrix $\mathcal{P}_{b,\alpha}$ expresses the basis b of $M = A$ in the basis of the decomposed module $M' = \bigoplus_{k=0}^{n-1} \mathbb{C}[x]/(x - \alpha_k)$. Thus, the representation ρ afforded by M' is given by $\rho = \phi^{\mathcal{P}_{b,\alpha}^{-1}}$. Since all the $\mathbb{C}[x]/(x - \alpha_k)$ are submodules (of dimension 1) of M, ρ is diagonal. The projection of $q(x) \in A$ onto $\mathbb{C}[x]/(x - \alpha_k)$ is just the evaluation $q(\alpha_k)$. Since for $q(x) = x$ all eigenvalues of $\phi(x)$ are distinct, all decomposition

matrices are given by $\mathcal{P}_{b,\alpha}^{-1} \cdot D$, D diagonal (and invertible). This shows (i). Part (ii) follows from (i) by transposition. □

Remark. The representation ϕ^T arises from the *right* regular module A.

Example 3.3 (Vandermonde matrix). Let $A = M = \mathbb{C}[x]/p(x)$, with separable p as above. We consider the special case $b = (x^0, x^1, \ldots, x^{n-1})$. Then the polynomial transform

$$\mathcal{P}_{b,\alpha} = \left[\alpha_k^\ell\right]_{k,\ell=0,\ldots,n-1}$$

is precisely the transpose of a Vandermonde matrix [5].

Next, we construct the regular representation ϕ of A with respect to the basis b. Since A is cyclic (generated by the polynomial x), it is sufficient to determine the image of $x \in A$ under ϕ. Let $p(x) = \sum_{i=0}^{n} \eta_i \cdot x^i$. Then

$$\begin{aligned} x \cdot x^i &= x^{i+1}, \quad i = 0, \ldots, n-2, \text{ and} \\ x \cdot x^{n-1} &= x^n \equiv \sum_{i=0}^{n-1} -\eta_i \cdot x^i \bmod p(x). \end{aligned}$$

Thus we obtain

$$\phi(x) = \begin{bmatrix} 0 & & & & -\eta_0 \\ 1 & 0 & & & -\eta_1 \\ & \ddots & \ddots & & \vdots \\ & & 1 & 0 & -\eta_{n-2} \\ & & & 1 & -\eta_{n-1} \end{bmatrix},$$

which is the transpose of the companion matrix of p. Using Lemma 3.2,

$$\phi(x)^{\mathcal{P}_{b,\alpha}^{-1}} = (\phi(x)^T)^{\mathcal{P}_{b,\alpha}^T} = \operatorname{diag}(\alpha_0, \ldots, \alpha_{n-1}).$$

Example 3.4 (discrete Fourier transform). We expand upon Example 3.3 by requiring also that $p(x) = x^n - 1$, which implies that $\alpha_k = e^{2\pi i k/n}$, $k = 0, \ldots, n-1$. In this case the transposed Vandermonde matrix coincides with the DFT of size n,

$$\mathrm{DFT}_n = \left[e^{2\pi i k\ell/n}\right]_{k,\ell=0,\ldots,n-1}.$$

This identifies the DFT as a polynomial transform. The corresponding representation ϕ maps x to the cyclic shift,

$$\phi(x) = \begin{bmatrix} 0 & & & 1 \\ 1 & 0 & & 0 \\ & \ddots & \ddots & \vdots \\ & & 1 & 0 \end{bmatrix},$$

and, by Lemma 3.2 and since DFT_n is symmetric,

$$(\phi(x)^T)^{\mathrm{DFT}(n)} = \operatorname{diag}_{k=0}^{n-1}(e^{2\pi i k/n}).$$

3.2. Scaled polynomial transforms. In section 3.1 we introduced polynomial transforms as Wedderburn matrices of regular A-modules M, where $A = M = \mathbb{C}[x]$. To capture all DTTs in an algebraic framework, we must generalize slightly the notion of polynomial transforms. In short, we will consider *scaled* polynomial transforms. These arise when the polynomials p_ℓ in (3.2) are replaced by $f \cdot p_\ell$, where f is a complex-valued function. Every scaled polynomial transform is associated with a regular module $M \cong A$, where M can be $\neq A$. We start with the following definition.

DEFINITION 3.5 (scaled polynomial transforms). *Let $\mathbb{C}[x]/p(x)$, b, and α be as in Definition* 3.1. *Further, let f be a complex-valued function satisfying $f(\alpha_k) \neq 0$, $k = 0, \ldots, n-1$. We define the* scaled *polynomial transform w.r.t. the scaling function f, basis b, and sample points α as*

$$\mathcal{P}_{f \cdot b, \alpha} = [(f \cdot p_\ell)(\alpha_k)]_{k,\ell=0,\ldots,n-1}. \tag{3.3}$$

We can associate a regular module to a scaled polynomial transform $\mathcal{P}_{f \cdot b, \alpha}$ in the following way. The vector space $f \cdot \mathbb{C}[x] = \{f \cdot q \mid q \in \mathbb{C}[x]\}$ naturally becomes a $\mathbb{C}[x]$-module by defining

$$r \cdot (f \cdot q) = f \cdot rq \quad \text{for } r \in \mathbb{C}[x].$$

Let p be a separable polynomial with zeros $\alpha = (\alpha_0, \ldots, \alpha_{n-1})$, and $A = \mathbb{C}[x]/p$. Then $\mathbb{C}[x] \cdot (f \cdot p)$ is a $\mathbb{C}[x]$-submodule of $f \cdot \mathbb{C}[x]$, and we can construct the factor module $M = f \cdot \mathbb{C}[x]/(\mathbb{C}[x] \cdot (f \cdot p)) = f \cdot \mathbb{C}[x]/(f \cdot p)$. We will briefly write $M = f \cdot A$. Its $\mathrm{ann}_{\mathbb{C}[x]}(M) = \mathbb{C}[x] \cdot p$, and thus M is an A-module, and if $b = (p_0, \ldots, p_{n-1})$ is a basis of A, then $f \cdot b = (f \cdot p_0, \ldots, f \cdot p_{n-1})$ is a basis of M.

We summarize the properties of the module $M = f \cdot A$ and the scaled polynomial transform $\mathcal{P}_{f \cdot b, \alpha}$ in the following lemma.

LEMMA 3.6. *Let $A = \mathbb{C}[x]/p(x)$, $b = (p_0, \ldots, p_{n-1})$ be a basis of A, and p be a separable polynomial with zeros $\alpha = (\alpha_0, \ldots, \alpha_{n-1})$. Assume that f is defined as above, and that $f(\alpha_k) \neq 0$, $k = 0, \ldots, n-1$. Further, let $M = f \cdot A$ with basis $f \cdot b$ as defined above. Then the following hold:*

(i) *M is a regular A-module.*

(ii) *The (regular) representation ϕ of A afforded by M and $f \cdot b$ is equal to the (regular) representation of A afforded by A and b.*

(iii) $\mathcal{P}_{f \cdot b, \alpha} = \mathrm{diag}(f(\alpha_0), \ldots, f(\alpha_{n-1})) \cdot \mathcal{P}_{b, \alpha}$.

(iv) *The representation ϕ is diagonalized by $\mathcal{P}^{-1}_{f \cdot b, \alpha}$, and the representation ϕ^T is diagonalized by $\mathcal{P}^T_{f \cdot b, \alpha}$.*

Proof. (i) follows, since $f \not\equiv 0$, $p_i \mapsto f \cdot p_i$, $i = 0, \ldots, n-1$, defines an isomorphism $A \to f \cdot A$. (ii) is obvious. (iii) follows straight from the definitions in (3.2) and (3.3). (iv) follows from (iii) and Lemma 3.2. ∎

Remark. The scaled polynomial transform $\mathcal{P}_{f \cdot b, \alpha}$ is not the Wedderburn matrix of the module $f \cdot A$ with basis $f \cdot b$. The mapping $f \cdot p_k \mapsto p_k$, $k = 0, \ldots, n-1$, defines an A module isomorphism between $f \cdot A$ and A. The corresponding matrix (w.r.t. the bases $f \cdot b$ and b) is the identity. Hence $f \cdot A$ and A have the same Wedderburn matrix $\mathcal{P}_{b, \alpha}$.

4. Chebyshev polynomials. In this section we introduce a general class of Chebyshev polynomials and their properties that we will use throughout this paper. We start with the classical cases.

TABLE 4.1
T_n and U_n for $-2 \le n \le 3$.

n	-2	-1	0	1	2	3
T_n	$2x^2-1$	x	1	x	$2x^2-1$	$4x^3-3x$
U_n	-1	0	1	$2x$	$4x^2-1$	$8x^3-4x$

4.1. The classical cases. The classical Chebyshev polynomials (of the first kind) T_n are given by the three-term recurrence

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), \ n \ge 2. \tag{4.1}$$

T_n is a polynomial of degree n and can be written in a closed form as

$$T_n(x) = \cos n\theta, \quad \cos\theta = x \quad \text{for } x \in (-1,1). \tag{4.2}$$

The recurrence formula in (4.1) is symmetric and can also be run in the other direction to obtain Chebyshev polynomials with negative n. Doing this, we obtain the symmetry property $T_{-n} = T_n$, as can also be seen from (4.2). The sequence $\{T_n \mid n \ge 0\}$ is orthogonal on the interval $(-1,1)$ w.r.t. the weight function $w(x) = (1-x^2)^{-1/2}$, i.e.,

$$\int_{-1}^{1} T_n(x)T_m(x)w(x)dx = 0 \quad \text{for } n \ne m.$$

From the closed form (4.2) for T_n, we also readily read off its zeros as

$$\cos\frac{(k+1/2)\pi}{n}, \quad k = 0, \dots, n-1.$$

Using recurrence (4.1) with changed initial conditions yields the Chebyshev polynomials U_n of the second kind:

$$U_0(x) = 1, \quad U_1(x) = 2x, \quad U_n(x) = 2xU_{n-1}(x) - U_{n-2}(x), \ n \ge 2.$$

The closed form of U_n is given by

$$U_n(x) = \frac{\sin(n+1)\theta}{\sin\theta}, \quad \cos\theta = x \quad \text{for } x \in (-1,1),$$

and we get $U_{-1} = 0$ and the symmetry $U_{-n-2} = -U_n$. For $-2 \le n \le 3$ the polynomials T_n, U_n are shown in Table 4.1.

A thorough introduction to Chebyshev polynomials and orthogonal polynomials in general can be found in the books of Chihara [8], Szegö [48], and Rivlin [42].

4.2. Generalized Chebyshev polynomials. Now we consider the set $\mathcal{C}$ of all polynomial sequences $\{P_n \mid n \in \mathbb{Z}\}$ that satisfy the three-term recurrence

$$P_n(x) = 2xP_{n-1}(x) - P_{n-2}(x). \tag{4.3}$$

We will refer to each such sequence as Chebyshev polynomials.

LEMMA 4.1. *Let $\{P_n \mid n \in \mathbb{Z}\}$ be a sequence of Chebyshev polynomials (we drop the argument x for simplicity). Then the following hold:*

(i) $P_n = P_1 \cdot U_{n-1} - P_0 \cdot U_{n-2}$.
(ii) $T_k \cdot P_n = (P_{n+k} + P_{n-k})/2$.

TABLE 4.2
Four series of Chebyshev polynomials. The range for the zeros is $k = 0, \dots, n-1$.

	$n = 0, 1$	Closed form	Symmetry	Zeros	Weight $w(x)$
T_n	$1, x$	$\cos(n\theta)$	$T_{-n} = T_n$	$\cos \frac{(k+\frac{1}{2})\pi}{n}$	$\frac{1}{(1-x^2)^{1/2}}$
U_n	$1, 2x$	$\frac{\sin(n+1)\theta}{\sin\theta}$	$U_{-n} = -U_{n-2}$	$\cos \frac{(k+1)\pi}{n+1}$	$(1-x^2)^{1/2}$
V_n	$1, 2x-1$	$\frac{\cos(n+\frac{1}{2})\theta}{\cos\frac{1}{2}\theta}$	$V_{-n} = V_{n-1}$	$\cos \frac{(k+\frac{1}{2})\pi}{n+\frac{1}{2}}$	$\frac{(1+x)^{1/2}}{(1-x)^{1/2}}$
W_n	$1, 2x+1$	$\frac{\sin(n+\frac{1}{2})\theta}{\sin\frac{1}{2}\theta}$	$W_{-n} = -W_{n-1}$	$\cos \frac{(k+1)\pi}{n+\frac{1}{2}}$	$\frac{(1-x)^{1/2}}{(1+x)^{1/2}}$

Proof. Clearly, a sequence P_n is uniquely determined by its initial conditions P_0 and P_1. If P_0, P_1 give rise to the sequence P_n, and if Q is any polynomial, then $Q \cdot P_0, Q \cdot P_1$ give rise to the sequence $Q \cdot P_n$. If, further, P_0', P_1' give rise to the sequence P_n', then $P_0 + P_0', P_1 + P_1'$ give rise to $P_n + P_n'$.

(i) First we consider the initial polynomials $0, 1$ and $1, 0$ and obtain (cf. Table 4.1)

$$\begin{aligned} P_0 = 1,\ P_1 = 0: &\quad P_n = -U_{n-2}, \\ P_0 = 0,\ P_1 = 1: &\quad P_n = U_{n-1}. \end{aligned}$$

With the previous remark, this shows (i).

(ii) Induction on k. For $k = 0$ it is trivial; for $k = 1$ this is the defining recurrence (4.3) for P_n. Further, we have

$$\begin{aligned} T_{k+1} \cdot P_n &= (2xT_k - T_{k-1}) \cdot P_n \\ &= 2x \cdot (P_{n+k} + P_{n-k})/2 - (P_{n+k-1} + P_{n-k+1})/2 \\ &= (P_{n+k+1} + P_{n-k-1})/2, \end{aligned}$$

where we used the induction hypothesis in the second step, and (4.3) in the last step. □

Remarks. (1) Both assertions in Lemma 4.1 remain valid when the polynomials P_n and hence P_0, P_1 are allowed to be arbitrary complex-valued functions. (2) Lemma 4.1 shows that $\mathcal{C}$ is a free $\mathbb{C}[x]$-module of rank 2.

We are particularly interested in four polynomial sequences, T_n, U_n, V_n, W_n, in $\mathcal{C}$ arising from different initial conditions. The sequences T_n and U_n are the Chebyshev polynomials of the first and second kind, as introduced above. All four sequences can be written in a closed form, have simple symmetry properties, and are orthogonal on $(-1, 1)$ w.r.t. some weight function $w(x)$. The zeros in all cases can be obtained from the closed form. These properties are summarized in Table 4.2. The results on V_n and W_n can be found in [8, pp. 37, 39].

Later we will need the following arithmetic properties of the Chebyshev polynomials.

LEMMA 4.2. *The following hold for all $m, n \in \mathbb{Z}$:*

(i) $T_{mn} = T_n(T_m) = T_m(T_n)$.
(ii) $U_{mn-1} = U_{m-1}(T_n)U_{n-1}$.
(iii) $U_{2m} = V_m \cdot W_m$.

TABLE 4.3
Identities among the four series of Chebyshev polynomials; P_n has to be replaced by T_n, U_n, V_n, W_n to obtain rows $1, 2, 3, 4$, respectively.

	$P_n - P_{n-2}$	P_n	$P_n - P_{n-1}$	$P_n + P_{n-1}$
T_n	$2(x^2-1)U_{n-2}$	T_n	$(x-1)W_{n-1}$	$(x+1)V_{n-1}$
U_n	$2T_n$	U_n	V_n	W_n
V_n	$2(x-1)W_{n-1}$	V_n	$2(x-1)U_{n-1}$	$2T_n$
W_n	$2(x+1)V_{n-1}$	W_n	$2T_n$	$2(x+1)U_{n-1}$

(iv) $W_n(x) = (-1)^n V_n(-x)$.
(v) $T_n(1) = 1$.

Proof. The proof follows from the closed form of the polynomials (Table 4.2) and trigonometric identities. □

We conclude this section by stating an interesting property of the four types of Chebyshev polynomials introduced. Let P_n be any of T_n, U_n, V_n, W_n. Then, using well-known trigonometric identities, $P_n - P_{n-2}, P_n - P_{n-1}, P_n + P_{n+1}$ can again be expressed using these polynomials. In particular, this allows us to determine their zeros using Table 4.2. The complete set of identities is given in Table 4.3. The second column is trivial and is introduced to make the table comply with later investigations. As an example, row 2, column 1 shows that $U_n - U_{n-2} = 2T_n$.

Remarks. (1) In a few places in the literature, e.g., in [30], the four series of Chebyshev polynomials occur together. (2) If we use the closed forms of T_n, U_n to extend their definitions to rational $n \in \mathbb{Q}$, we can write $V_n = T_{n+1/2}/T_{1/2}$ and $W_n = U_{n-1/2}/U_{-1/2}$. (3) For a complete overview on the factorization of T_n and U_n over $\mathbb{Q}$, see [41].

5. The 16 types of DTTs. The first discrete cosine transform was introduced by Ahmed, Natarajan, and Rao [1]. The complete set of all eight types of DCTs and DSTs, respectively, was first presented by Wang and Hunt [53]. We will sometimes refer to them together as DTTs. Each of the transforms is given by an $(n \times n)$-matrix M, $n \geq 0$, which multiplies to a signal vector a from the left, $a \mapsto M \cdot a$. As examples, we will use the symbol DCT-2 to refer to a DCT of type 2, and DST-7_n to refer to a DST of type 7 and size n. If an arbitrary trigonometric transform is addressed, we will write DTT or DTT$_n$. In this notation, the first DCT introduced was of type 2.

TABLE 5.1
Eight types of DCTs and DSTs, given for size n. The entry at row k and column ℓ is given for $k, \ell = 0, \ldots, n-1$.

	DCTs	DSTs
type 1	$\cos k\ell \frac{\pi}{n-1}$	$\sin(k+1)(\ell+1)\frac{\pi}{n+1}$
type 2	$\cos k(\ell+\frac{1}{2})\frac{\pi}{n}$	$\sin(k+1)(\ell+\frac{1}{2})\frac{\pi}{n}$
type 3	$\cos(k+\frac{1}{2})\ell\frac{\pi}{n}$	$\sin(k+\frac{1}{2})(\ell+1)\frac{\pi}{n}$
type 4	$\cos(k+\frac{1}{2})(\ell+\frac{1}{2})\frac{\pi}{n}$	$\sin(k+\frac{1}{2})(\ell+\frac{1}{2})\frac{\pi}{n}$
type 5	$\cos k\ell \frac{\pi}{n-\frac{1}{2}}$	$\sin(k+1)(\ell+1)\frac{\pi}{n+\frac{1}{2}}$
type 6	$\cos k(\ell+\frac{1}{2})\frac{\pi}{n-\frac{1}{2}}$	$\sin(k+1)(\ell+\frac{1}{2})\frac{\pi}{n+\frac{1}{2}}$
type 7	$\cos(k+\frac{1}{2})\ell\frac{\pi}{n-\frac{1}{2}}$	$\sin(k+\frac{1}{2})(\ell+1)\frac{\pi}{n+\frac{1}{2}}$
type 8	$\cos(k+\frac{1}{2})(\ell+\frac{1}{2})\frac{\pi}{n+\frac{1}{2}}$	$\sin(k+\frac{1}{2})(\ell+\frac{1}{2})\frac{\pi}{n-\frac{1}{2}}$

Table 5.1 gives the definitions of all 16 types of DCTs and DSTs by stating the

respective entry at position (k, ℓ), where k is the row index, for $k, \ell = 0, \ldots, n-1$. As can be seen, all entries are pure cosines or sines of the form $\cos r\pi$ or $\sin r\pi$, where r is some rational number. Thus, all entries are elements in a suitable cyclotomic field over $\mathbb{Q}$. The definitions given in Table 5.1 are the *unscaled* versions of the DCTs and DSTs, which will be considered in this paper. The *scaled* versions of the DCTs and DSTs are orthonormal and arise from the unscaled versions by multiplying in some cases the first and/or last row and/or column by $1/\sqrt{2}$, which makes the matrix orthogonal. In addition, the entire matrix is multiplied by a factor to achieve orthonormality. As an example, the orthonormal version of the DCT-2 has entries

$$\sqrt{\frac{2}{n}} \cdot c_k \cdot \cos k \left(\ell + \frac{1}{2}\right) \frac{\pi}{n}, \quad k, \ell = 0, \ldots, n-1,$$

where $c_k = \sqrt{1/2}$ for $k = 0$ and $c_k = 1$ elsewhere. For the convenience of the reader, the scaled, orthonormal versions of the DCTs and DSTs are given in Table A.1 in the appendix. Note that the set of all DTTs is closed under matrix transposition. From Table 5.1 it is easily seen that the DCT and the DST of types 1, 4, 5, and 8 are symmetric, and that types 2 and 3 and types 6 and 7 are converted into each other, respectively, by transposition.

All 16 DCTs and DSTs arise as eigenmatrices of certain tridiagonal matrices [47, 45] of size $(n \times n)$, which can be chosen of the form

$$B(\beta_1, \beta_2, \beta_3, \beta_4) = \frac{1}{2} \cdot \begin{bmatrix} \beta_1 & \beta_2 & & & & & \\ 1 & 0 & 1 & & & & \\ & 1 & 0 & 1 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & 1 & 0 & 1 & \\ & & & & \beta_3 & \beta_4 & \end{bmatrix}. \tag{5.1}$$

The internal structure of $B(\beta_1, \beta_2, \beta_3, \beta_4)$ corresponds to the equation

$$a_k = \frac{1}{2}(a_{k-1} + a_{k+1}), \quad 1 \le k \le n-2. \tag{5.2}$$

The entries β_1, β_2 are determined by a choice of *left boundary conditions (b.c.)* that determine how a_{-1} is chosen in (5.2) for $k = 0$. The four left b.c. considered are $a_{-1} = a_1$, $a_{-1} = 0$, $a_{-1} = a_0$, $a_{-1} = -a_0$. For example, the choice $a_{-1} = a_1$ leads to $\beta_1 = 0, \beta_2 = 2$. Similarly, the entries β_3, β_4 are determined by right b.c. arising from the choice of a_n in (5.2) for $k = n-1$. The right b.c. are the mirrored versions of the left b.c.: $a_n = a_{n-2}$, $a_n = 0$, $a_n = a_{n-1}$, $a_n = -a_{n-1}$. The complete set of values $\beta_1, \beta_2, \beta_3, \beta_4$ for all 16 possible combinations of b.c. is given in Table 5.2. If b.c., and thus values $\beta_1, \beta_2, \beta_3, \beta_4$, are chosen, and if $a = (a_0, \ldots, a_{n-1})^T$, then (5.2), $k = 0, \ldots, n-1$, can be written as

$$a = B(\beta_1, \beta_2, \beta_3, \beta_4) \cdot a.$$

Remarks. (1) The matrices $B(\cdot)$ in (5.1) correspond to linear time-invariant processes with b.c. [33, 47]. (2) The b.c. $a_{-1} = 0$ and $a_{-1} = -a_0$ are the discrete versions of Dirichlet b.c.; $a_{-1} = a_1$ and $a_{-1} = a_0$ are the discrete versions of Neumann b.c. Analogously for the right b.c. [33, 47].

The 16 DTTs correspond to these different choices of b.c. as shown in Table 5.3 (see [47]). The relationship is as follows. If numbers $\beta_1, \beta_2, \beta_3, \beta_4$ (and hence left and

TABLE 5.2
The values $\beta_1, \beta_2, \beta_3, \beta_4$ from (5.1) for the four respective choices of left b.c. and right b.c.

Left b.c.	β_1	β_2
$a_{-1} = a_1$	0	2
$a_{-1} = 0$	0	1
$a_{-1} = a_0$	1	1
$a_{-1} = -a_0$	−1	1

Right b.c.	β_3	β_4
$a_n = a_{n-2}$	2	0
$a_n = 0$	1	0
$a_n = a_{n-1}$	1	1
$a_n = -a_{n-1}$	1	−1

TABLE 5.3
The left and right boundary conditions associated with the DCTs and DSTs.

	$a_n = a_{n-2}$	$a_n = 0$	$a_n = a_{n-1}$	$a_n = -a_{n-1}$
$a_{-1} = a_1$	DCT-1	DCT-3	DCT-5	DCT-7
$a_{-1} = 0$	DST-3	DST-1	DST-7	DST-5
$a_{-1} = a_0$	DCT-6	DCT-8	DCT-2	DCT-4
$a_{-1} = -a_0$	DST-8	DST-6	DST-4	DST-2

right b.c.) are chosen from row k and row ℓ, respectively, of Table 5.2 ($k, \ell = 1, \ldots, 4$), then the corresponding matrix $B(\beta_1, \beta_2, \beta_3, \beta_4)$ is diagonalized by *the transpose* of the DTT of size n given in row k and column ℓ of Table 5.3.

Example 5.1. As an example, we choose left b.c. $a_{-1} = a_0$ and right b.c. $a_n = a_{n-1}$ and obtain $\beta_1 = \beta_2 = \beta_3 = \beta_4 = 1$. The $(n \times n)$-matrix

$$B(1,1,1,1) = \frac{1}{2} \cdot \begin{bmatrix} 1 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & \cdot & \cdot & \cdot & & \\ & & 1 & 0 & 1 & \\ & & & 1 & 1 \end{bmatrix} \tag{5.3}$$

is diagonalized by DCT-2_n^T = DCT-3_n; i.e., $B(1,1,1,1)^{\text{DCT-3}_n}$ is diagonal.

Remarks. (1) The DTTs of types 5–8 are also called "odd" DTTs of types 1–4, respectively. (2) Reference [47] considers the matrices $2\,\mathrm{I} - 2B(\cdot)$ rather than the matrices $B(\cdot)$, which leads to equivalent diagonalization properties. Also the definition of the DTTs is transposed to our definition. We chose the original [53] and commonly used definition.

6. The algebraic characterization of the DTTs. In this section we will show that all 16 DTTs are scaled polynomial transforms (see section 5) by constructing the corresponding modules and bases. To connect, for a given DTT, its diagonalization property, i.e., the associated matrix $B(\beta_1, \beta_2, \beta_3, \beta_4)$ (cf. section 5), with the algebra/module framework, we will construct a module with basis b that affords a representation ϕ such that

$$\phi^T(x) = B(\beta_1, \beta_2, \beta_3, \beta_4).$$

In other words, the operation of x (via multiplication) on b is reflected by the matrix $B(\beta_1, \beta_2, \beta_3, \beta_4)$. Lemma 3.6(iv) will establish the correspondence between the DTT and the module constructed this way.

The construction of the module and its base is a three-step procedure. Assume a DTT and an associated matrix $B(\cdot)$ are given.

1. *Internal structure* (section 6.1). Determine a sequence of polynomials that yields the internal structure of $B(\cdot)$, i.e., the $\ldots, \frac{1}{2}, 0, \frac{1}{2}, \ldots$ in each column. This will bring into play generalized Chebyshev polynomials in a natural way.

2. *Left boundary conditions* (section 6.2). Fix the left b.c. This corresponds to fixing the initial conditions for the Chebyshev polynomials, i.e., the choice of a particular sequence of Chebyshev polynomials.

3. *Right boundary conditions* (section 6.3). Fix the right b.c. This corresponds to choosing the appropriate polynomial p for the module (and the algebra) $\mathbb{C}[x]/p$.

6.1. Internal structure. First we will consider n-dimensional modules that carry the structure given in (5.2). Rewriting (4.3) in a slightly different form as

$$x \cdot P_k = \frac{1}{2}(P_{k-1} + P_{k+1}) \tag{6.1}$$

shows that this is afforded by any regular module $A = \mathbb{C}[x]/p$, $\deg(p) = n$, if we choose the basis $b = (P_0, \ldots, P_{n-1})$, where the P_k are generalized Chebyshev polynomials. In other words, the image of x under the representation ϕ afforded by A with basis b will have an internal structure similar to the matrices given in (5.1).

6.2. Left boundary conditions. The four left b.c. associated with the DTTs are (see Table 5.3)

$$a_{-1} = a_1, \quad a_{-1} = 0, \quad a_{-1} = a_0, \quad a_{-1} = -a_0. \tag{6.2}$$

They apply in the boundary case $k = 0$ in (5.2). An equivalent behavior is obtained in (6.1) if we choose the four special sequences of Chebyshev polynomials T_k, U_k, V_k, W_k introduced in Table 4.2. The symmetry properties of these polynomials (cf. Table 4.2) correspond to the left b.c. in (6.2),

$$T_{-1} = T_1, \quad U_{-1} = 0, \quad V_{-1} = V_0, \quad W_{-1} = -W_0,$$

respectively. As an example, *every* regular module $\mathbb{C}[x]/p$ with basis $(T_0, \ldots, T_{n-1})$ carries the left b.c. $a_{-1} = a_1$.

6.3. Right boundary conditions. The four right b.c. associated with the DTTs mirror the left b.c. (see Table 5.3):

$$a_n = a_{n-2}, \quad a_n = 0, \quad a_n = a_{n-1}, \quad a_n = -a_{n-1}. \tag{6.3}$$

The right b.c. are determined by the choice of p in $\mathbb{C}[x]/p$. As an example, to introduce the right b.c. $a_n = a_{n-2}$, we choose $p = P_n - P_{n-2}$, where $P \in \{T, U, V, W\}$. Thus the choices of p corresponding to (6.3) are

$$P_n - P_{n-2}, \quad P_n, \quad P_n - P_{n-1}, \quad P_n + P_{n-1}, \tag{6.4}$$

respectively. To determine the zeros of p in these cases, and hence the decomposition of A and its associated decomposing polynomial transform, we need to consult Table 4.3, which covers all cases in (6.4) for $P \in \{T, U, V, W, \}$.

6.4. Summary. Before we state the interpretation of the DTTs as scaled polynomial transforms, it is perhaps instructive to consider an example.

Example 6.1 (DST-3). We choose the left b.c. $a_{-1} = 0$, which leads to the choice of the basis $b = (U_0, \ldots, U_{n-1})$. As right b.c. we choose $a_n = a_{n-2}$, which leads to

$p = U_n - U_{n-2} = 2T_n$ using Table 4.3. The decomposition of the regular module $A = \mathbb{C}[x]/T_n$ (the 2 can be dropped) is determined by the zeros of T_n, which are $\alpha = (\cos \frac{1}{2}\pi/n, \ldots, \cos(n - \frac{1}{2})\pi/n)$ (cf. Table 4.2), i.e.,

$$A = \mathbb{C}[x]/(U_n - U_{n-2}) = \mathbb{C}[x]/T_n = \bigoplus_{k=0}^{n-1} \mathbb{C}[x]/\left(x - \cos\left(k + \tfrac{1}{2}\right)\pi/n\right).$$

The decomposing polynomial transform is given by

$$\begin{aligned}\mathcal{P}_{b,\alpha} &= [U_\ell(\cos(k+1/2)\pi/n)]_{k,\ell=0,\ldots,n-1} \\ &= \left[\frac{\sin(\ell+1)(k+1/2)\pi/n}{\sin(k+1/2)\pi/n}\right]_{k,\ell=0,\ldots,n-1} \\ &= \operatorname{diag}_{k=0}^{n-1}\left(\frac{1}{\sin(k+1/2)\pi/n}\right) \cdot \text{DST-3}_n,\end{aligned}$$

which shows that DST-3_n is the scaled polynomial transform

$$\text{DST-3}_n = \mathcal{P}_{f\cdot b,\alpha}, \quad f = \sin\theta,$$

associated with the module $f \cdot A$ with basis $f \cdot b$.

Next we construct the representation ϕ afforded by A with basis b. By construction, we have $x \cdot U_0 = \frac{1}{2}U_1$, $x \cdot U_\ell = \frac{1}{2}(U_{\ell-1} + U_{\ell+1})$ for $\ell = 1, \ldots, n-2$, and $x \cdot U_{n-1} = U_{n-2}$ (in A). We get

$$\phi(x) = \frac{1}{2} \cdot \begin{bmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & \cdot & \cdot & \cdot & & \\ & & 1 & 0 & 2 & \\ & & & 1 & 0 & \end{bmatrix}.$$

Lemma 3.6 shows that $\phi(x)^T$ is diagonalized by DST-3_n^T = DST-2_n, namely,

$$\phi^T(x)^{\text{DST-2}_n} = \operatorname{diag}\left(\cos\tfrac{1}{2}\pi/n, \ldots, \cos\left(n - \tfrac{1}{2}\right)\pi/n\right).$$

Using the notation from section 5, $\phi(x)^T = B(0, 1, 2, 0)$, which corresponds to the DST-3 (see Tables 5.2 and 5.3), as desired.

Remarks. (1) It is intriguing that the left and right b.c. seem to be handled differently (initial conditions versus factor polynomial). In section 8.1 we will see that this construction can be reversed. (2) Note that the boundary conditions corresponding to the *left* module constructed affect the first and last *column* of the *left* representation $\phi(x)$. Lemma 3.2 shows that the *right* representation ϕ^T is decomposed by the transpose of the corresponding DTT, which complies with the fact that the b.c. affect the first and last *rows* in the matrices $B(\cdot)$ (cf. section 5). (3) The polynomial defining the right b.c. in Example 6.1 can be written in two ways, $U_n - U_{n-2} = 2T_n$ (cf. Table 4.3). The left form determines the b.c.; the right form provides the decomposition of $\mathbb{C}[x]/T_n$, which corresponds to the zeros of T_n.

The complete correspondence between DTTs and modules is given in Theorem 6.2 below. To provide a convenient overview, and because we will repeatedly use it in the following, we have combined Tables 4.3 and 5.3 and the respective scaling functions into Table 6.1.

TABLE 6.1

Overview of the DTTs and associated modules. The left b.c. and right b.c. are in the first column (value of a_{-1}) and row, respectively. A given DTT_n is associated with the module $f \cdot \mathbb{C}[x]/Q_n$, where Q_n is given below the DTT and the scaling function f in the second column. The basis of $\mathbb{C}[x]/Q_n$ is given in the third column.

			$a_n - a_{n-2}$	a_n	$a_n - a_{n-1}$	$a_n + a_{n-1}$
a_1	1	T_ℓ	DCT-1 $2(x^2-1)U_{n-2}$	DCT-3 T_n	DCT-5 $(x-1)W_{n-1}$	DCT-7 $(x+1)V_{n-1}$
0	$\sin\theta$	U_ℓ	DST-3 $2T_n$	DST-1 U_n	DST-7 V_n	DST-5 W_n
a_0	$\cos\frac{1}{2}\theta$	V_ℓ	DCT-6 $2(x-1)W_{n-1}$	DCT-8 V_n	DCT-2 $2(x-1)U_{n-1}$	DCT-4 $2T_n$
$-a_0$	$\sin\frac{1}{2}\theta$	W_ℓ	DST-8 $2(x+1)V_{n-1}$	DST-6 W_n	DST-4 $2T_n$	DST-2 $2(x+1)U_{n-1}$

THEOREM 6.2. *Define the four scaling functions $f_1 = 1$, $f_2 = \sin\theta$, $f_3 = \cos\frac{1}{2}\theta$, and $f_4 = \sin\frac{1}{2}\theta$, with $\cos\theta = x$. Choose a combination of left and right boundary conditions with index i, j from Table 6.1, $i, j = 1, \ldots, 4$, and let DTT_n be the corresponding discrete trigonometric transform. Denote the polynomial below the DTT in Table 6.1 by Q_n and its zeros by $\alpha = (\alpha_0, \ldots, \alpha_{n-1})$. Choose a basis of $A = \mathbb{C}[x]/Q_n$ as $b = (P_0, \ldots, P_{n-1})$, where $P = T, U, V, W$ for $i = 1, 2, 3, 4$, respectively. Then*

(i) *DTT_n is the scaled polynomial transform*

$$\mathrm{DTT}_n = \mathcal{P}_{f_i \cdot b, \alpha}$$

associated with the module $f_i \cdot A$ and basis $f_i \cdot b$.

(ii) *If ϕ is the representation afforded by A with b, then $\phi(x)^T$ is the matrix $B(\cdot)$ in (5.1) given by the left and right b.c. chosen.*

(iii) *The matrix $\phi(x)^T$ is diagonalized by DTT_n^T, namely,*

$$(\mathrm{DTT}_n^T)^{-1} \cdot \phi(x)^T \cdot \mathrm{DTT}_n^T = \mathrm{diag}(\alpha_0, \ldots, \alpha_{n-1}),$$

which implies that DTT_n^T is a decomposition matrix for the (right) regular representation ϕ^T of A.

Proof. The proof follows by computations completely analogous to Example 6.1 for all 16 cases. ∎

Remarks. (1) Theorem 6.2 shows that a DTT is a polynomial transform (i.e., not scaled) iff it appears in the first row of Table 6.1. For the DCT-1 and the DCT-3 this has been recognized in [35] and [46], respectively. (2) The sparse matrices $B(\cdot)$ occur as images of $T_1 = x$ under the (right) representation ϕ^T of the respective module. Using Lemma 4.1(ii), one can compute the images $\phi^T(T_k)$, $k = 0, \ldots, n-1$, which all turn out to be sparse. This makes $(T_0, \ldots, T_{n-1})$ a natural choice of basis in the algebra (not the module) A in all 16 cases. The image $\phi^T(a)$ (or $\phi(a)$) of a generic element $a = \sum a_k T_k \in A$ has a structure that is usually referred to as Toeplitz + Hankel.

With the algebraic characterization of the DTTs given in Theorem 6.2, we are now in the position to derive and explain many of their fast algorithms known from the literature. This is the subject of the remaining sections.

7. Fast algorithms for polynomial transforms. Fast algorithms for the matrix-vector multiplication with polynomial transforms, $z \mapsto \mathcal{P}_{b,\alpha} \cdot z$ or, equivalently, sparse factorizations of $\mathcal{P}_{b,\alpha}$, have been the subject of several papers. In [46]

the DCT-3 and the real and imaginary parts of the DFT are recognized as polynomial transforms, which is used for their factorization. In [14] and in [35] an $O(n \log^2 n)$ algorithm is derived for the case in which b is an arbitrary sequence of orthogonal polynomials and α a list of arbitrary (distinct) evaluation points. Using this result in combination with Theorem 6.2 shows that the complexity of computing a DTT_n is $O(n \log^2 n)$. The fast DTT algorithms known from the literature, however, and the following discussion show that the complexity is indeed $O(n \log n)$.

In this section we will present two general techniques that can be used to factor a polynomial transform $\mathcal{P}_{b,\alpha}$ associated with the regular module $\mathbb{C}[x]/p$. They apply in the cases

1. $p(x) = q(x) \cdot r(x)$ (p factors),
2. $p(x) = q(r(x))$ (p decomposes).

An important question is to know when the resulting matrix factors are sparse, yielding a fast algorithm. We note that it is also possible to factor $\mathcal{P}_{b,\alpha}$ if

3. $p(x) = q(x) \otimes r(x)$ (p is a tensor product),

but we omit this case since it does not apply to the DTTs. Because of Lemma 3.6, the problems of finding fast algorithms for $\mathcal{P}_{b,\alpha}$ and $\mathcal{P}_{f \cdot b,\alpha}$ are equivalent.

Throughout this section, p is a separable polynomial with zero vector α.

7.1. Direct sum. One straightforward way of obtaining a fast polynomial transform is by splitting the polynomial p recursively, using the fact that, if $p = q \cdot r$,

$$\mathbb{C}[x]/p \cong \mathbb{C}[x]/q \oplus \mathbb{C}[x]/r. \tag{7.1}$$

This reduces the problem of computing one polynomial transform to the computation of two smaller polynomial transforms.

LEMMA 7.1. *Let $p = q \cdot r$, and assume that p, q, r have the zero vectors α, β, γ, respectively. Further, let b, c, d be bases of $\mathbb{C}[x]/p$, $\mathbb{C}[x]/q$, $\mathbb{C}[x]/r$, respectively. Then*

$$\mathcal{P}_{b,\alpha} = P \cdot (\mathcal{P}_{c,\beta} \oplus \mathcal{P}_{d,\gamma}) \cdot B,$$

where B is the base change matrix $b \to (c, d)$ (concatenation) corresponding to (7.1) *and P is a permutation matrix mapping $(\beta, \gamma) \mapsto \alpha$.*

Proof. The proof follows from the definitions of B and P. ∎

Clearly, the decomposition in Lemma 7.1 is useful for a fast algorithm only if B is sparse or has a fast algorithm itself. As an example, the fast algorithm for the Vandermonde matrix relies on the fact that in this case B has a Toeplitz structure, which permits its computation with $O(n \log n)$ arithmetic operations [14, 34].

7.2. Decomposition. A more interesting factorization of a polynomial transform can be derived if p *decomposes* into two polynomials, $p(x) = q(r(x))$. We will need the following lemma.

LEMMA 7.2. *Let p be separable and of degree n with zeros $\alpha_0, \ldots, \alpha_{n-1}$. Assume $p(x) = q(r(x))$ with q of degree k and r of degree ℓ. Then for each zero β of q there are precisely ℓ zeros α_m of p such that $r(\alpha_m) = \beta$.*

Proof. Let α_m be a zero of p. Then $0 = p(\alpha_m) = q(r(\alpha_m))$. Thus r maps the $n = k\ell$ zeros of p to the k zeros of q. If β is one of the k zeros of q, then the equation $r(\alpha_m) = \beta$ has maximal $\deg(r) = \ell$ solutions α_m; thus it has precisely ℓ solutions. ∎

As in Lemma 7.2, let the degrees of p, q, r be n, k, ℓ, respectively, $n = k\ell$. We

choose bases $c = (q_0, \dots, q_{k-1})$ of $\mathbb{C}[x]/q$ and $d = (r_0, \dots, r_{\ell-1})$ of $\mathbb{C}[x]/r$. Then

$$\begin{aligned} b' = (&r_0 \cdot q_0(r), \dots, r_0 \cdot q_{k-1}(r),\\ &r_1 \cdot q_0(r), \dots, r_1 \cdot q_{k-1}(r),\\ &\dots\dots\dots\dots\dots\dots\dots\dots\\ &r_{\ell-1} \cdot q_0(r), \dots, r_{\ell-1} \cdot q_{k-1}(r)) \end{aligned}$$

is a basis of $\mathbb{C}[x]/p$. Using the shorter notation $p_{j,i,m} = (r_j \cdot q_i(r))(\alpha_m)$, the corresponding polynomial transform is given by

$$\mathcal{P}_{b',\alpha} = \begin{bmatrix} p_{0,0,0} & \cdots & p_{0,k-1,0} & \cdots & p_{\ell-1,0,0} & \cdots & p_{\ell-1,k-1,0} \\ p_{0,0,1} & \cdots & p_{0,k-1,1} & \cdots & p_{\ell-1,0,1} & \cdots & p_{\ell-1,k-1,1} \\ \dots & & \dots & & \dots & & \dots \\ p_{0,0,n-1} & \cdots & p_{0,k-1,n-1} & \cdots & p_{\ell-1,0,n-1} & \cdots & p_{\ell-1,k-1,n-1} \end{bmatrix}.$$

Because of Lemma 7.2, for each i, the n numbers $q_i(r(\alpha_m))$, $m = 0, \dots, n-1$, will divide into k groups of ℓ equals. We permute α into α' with a permutation P such that $r(\alpha_{i+jk}) = \beta_i$, $i = 0, \dots, k-1$, $j = 0, \dots, \ell-1$, i.e.,

$$\mathcal{P}_{b',\alpha'} = P \cdot \mathcal{P}_{b',\alpha}.$$

Now $\mathcal{P}_{b',\alpha'}$ reveals the following block structure:

$$\begin{aligned} \mathcal{P}_{b',\alpha'} &= [D_{h,j} \cdot \mathcal{P}_{c,\beta}]_{h,j=0,\dots,\ell-1}, \quad \text{with} \\ D_{h,j} &= \operatorname{diag}(r_j(\alpha'_{hk}), r_j(\alpha'_{hk+1}), \dots, r_j(\alpha'_{hk+k-1})). \end{aligned}$$

Thus we can write $\mathcal{P}_{b',\alpha'}$ as

$$\mathcal{P}_{b',\alpha'} = [D_{h,j}]_{h,j=0,\dots,\ell-1} \cdot (\mathrm{I}_\ell \otimes \mathcal{P}_{c,\beta}).$$

Since $D_{h,j}$ is diagonal, $h, j = 0, \dots, \ell-1$, the matrix $[D_{h,j}]$ consists of k $(\ell \times \ell)$ blocks at stride k. Thus,

$$[D_{h,j}]^{\mathrm{L}^n_\ell}$$

is a direct sum of $(\ell \times \ell)$-matrices, which turn out to again be polynomial transforms. Using $(\mathrm{L}^n_\ell)^{-1} = \mathrm{L}^n_k$, we get the following theorem.

THEOREM 7.3. *We use the previous notation. Then*

$$\mathcal{P}_{b,\alpha} = P \cdot \left(\bigoplus_{i=0}^{k-1} \mathcal{P}_{d,\overline{\alpha}_i} \right)^{\mathrm{L}^n_k} \cdot (\mathrm{I}_\ell \otimes \mathcal{P}_{c,\beta}) \cdot B,$$

where B is the matrix giving the base change $b \to b'$, P is a permutation matrix, and

$$\overline{\alpha}_i = (\alpha'_{0 \cdot k+i}, \alpha'_{1 \cdot k+i}, \dots, \alpha'_{(\ell-1) \cdot k+i}).$$

As in Lemma 7.1, the value of this factorization for deriving a fast algorithm for $\mathcal{P}_{b,\alpha}$ depends on the base change matrix B.

Theorem 7.3 can be interpreted as a generalization of the Cooley–Tukey FFT as we will see in the next example.

Example 7.4 (FFT, size 4). We consider the case $p(x) = x^4 - 1 = (x^2)^2 - 1$, i.e., $q(x) = x^2 - 1$, and $r(x) = x^2$. As bases we choose $b = (1, x, x^2, x^3)$ and $c = d = (1, x)$.

The zeros of p are $\alpha = (1, i, -1, -i)$, and the zeros of q are $\beta = (1, -1)$. This is the situation of Example 3.4, $\mathcal{P}_{b,\alpha} = \mathrm{DFT}_4$, $\mathcal{P}_{c,\beta} = \mathrm{DFT}_2$. Since $r(1) = r(-1)$ and $r(i) = r(-i)$, it is $\alpha' = \alpha$. Further, $b' = (1, x^2, x, x^3)$, and thus $B = [(2, 3), 4] = \mathrm{L}^4_2$. It remains to compute $\mathcal{P}_{d,\overline{\alpha}_0}, \mathcal{P}_{d,\overline{\alpha}_1}$:

$$\mathcal{P}_{d,\overline{\alpha}_0} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \mathrm{DFT}_2, \qquad \mathcal{P}_{d,\overline{\alpha}_1} = \begin{bmatrix} 1 & i \\ 1 & -i \end{bmatrix} = \mathrm{DFT}_2 \cdot \mathrm{diag}(1, i).$$

As a result we get the FFT of size 4,

$$\mathrm{DFT}_4 = (\mathrm{DFT}_2 \otimes \mathrm{I}_2) \cdot \mathrm{diag}(1, 1, 1, i) \cdot (\mathrm{I}_1 \otimes \mathrm{DFT}_2) \cdot \mathrm{L}^4_2 .$$

Remark. It is worth giving an algebraic interpretation of Theorem 7.3 using the notation above. Since $p(x) = q(r(x))$, $A' = \mathbb{C}[r(x)]/p(x) = \mathbb{C}[y]/q(y)$ $(y = r(x))$ is a subalgebra of $A = C[x]/p(x)$. We have

$$A \cong r_0 \cdot A' \oplus \cdots \oplus r_{\ell-1} \cdot A'$$

as vector spaces; i.e., $d = (r_0, \ldots, r_{\ell-1})$ is a transversal of A' in A. In a similar way as was done for $\mathbb{C}[G]$-modules (G a group) [12, p. 73], we can construct the induced module

$$A \otimes_{A'} A' = (r_0 \otimes A') \oplus \cdots \oplus (r_{\ell-1} \otimes A'),$$

which has the basis b'. The modules A and $A \otimes'_A A'$ are isomorphic with the base change given by the matrix B. Thus, Theorem 7.3 (for polynomial algebras) is the equivalent of Theorem 3.33 (for group algebras of solvable groups) in [37]. They coincide for the case $\mathbb{C}[Z_n] \cong \mathbb{C}[x]/(x^n - 1)$ (Z_n = cyclic group of order n), where they yield the Cooley–Tukey FFT (cf. Example 7.4).

8. Fast DTTs via decomposition of polynomial transforms. In this section we derive and explain several different recursive algorithms for the DTTs directly from their algebraic interpretation. In contrast to the derivations given in the literature, we do not manipulate matrix entries; rather, we obtain the algorithm directly from the underlying modules. This makes the derivation simpler and more transparent, and provides a mathematically satisfying insight into the structure of the algorithm.

The algorithms presented in this section can be loosely grouped into the following categories:

1. *Translation* (section 8.1). A DTT is translated into another DTT using sparse matrices. Two different methods are identified.
2. *Direct sum* (section 8.2). A DTT is decomposed into the direct sum of smaller DTTs using sparse matrices. These algorithms are due to Lemma 7.1.
3. *Reduction* (section 8.3). A DTT is decomposed into smaller DTTs of the same type using sparse matrices. These algorithms are due to Theorem 7.3.

It is important to note that we can always derive, from any given fast algorithm, new fast algorithms by straightforward operations like symbolic transposition or inversion, since these are compatible with $\otimes$ and $\oplus$. As an example, a factorization like

$$\text{DCT-2}_n = P \cdot (\text{DCT-2}_{n/2} \oplus \text{DCT-4}_{n/2}) \cdot B$$

can be transposed to yield

$$\text{DCT-3}_n = B^T \cdot (\text{DCT-3}_{n/2} \oplus \text{DCT-4}_{n/2}) \cdot P^T,$$

since $\text{DCT-2}^T = \text{DCT-3}$, and DCT-4 is symmetric. Moreover, it is always possible to locally manipulate these formula expressions. As an example, let Q and R be permutations. Then

$$Q \cdot (\mathrm{I}_n \otimes \mathrm{DFT}_2) \cdot R = (QP^{-1}) \cdot (\mathrm{I}_n \otimes \mathrm{DFT}_2) \cdot (PR)$$

for any permutation P permuting (2×2)-blocks ($n!$ such P). We will consider algorithms that can be transformed into each other using manipulations of this kind as "algebraically equivalent." The comparisons between the algorithms we derive here and the algorithms from the literature have to be understood "modulo" this equivalence, though, in many cases, the comparison will be exact.

8.1. Translation between DTTs. In this section we will discuss and derive sparse relationships between the different types of DTTs. We say that DTT_n and DTT'_n are in sparse relationship if DTT_n can be derived from DTT'_n using $O(n)$ operations. An example of a sparse relationship is the equation

$$\mathrm{DTT}_n = B_n \cdot \mathrm{DTT}'_n \cdot C_n,$$

where B_n, C_n are sparse matrices ($O(n)$ entries). These relationships are important for fast algorithms. If a fast algorithm for DTT_n is given, and DTT_n and DTT'_n are in sparse relationship, then we obtain a fast algorithm for DTT'_n, and vice versa.

Examining Table 6.1, we observe that transform pairs at transposed positions (i, j) and (j, i), $i, j = 1, \ldots, 4$, have the same associated algebra (i.e., the same polynomial Q_n). As an example, DCT-5 and DCT-6 both arise from $\mathbb{C}[x]/(x-1)W_{n-1}$ with different bases. This leads to the concept of duality introduced in the next definition.

Definition 8.1 (duality). *We call a pair* DTT_n, DTT'_n *dual to each other if the left b.c. of* DTT_n *correspond to the right b.c. of* DTT'_n *and vice versa. Equivalently,* DTT_n *and* DTT'_n *appear in transposed positions* (i, j) *and* (j, i) *in Table* 6.1. *If* $i = j$, *we call* $\mathrm{DTT}_n = \mathrm{DTT}'_n$ *self-dual.*

We show how this duality can be used to derive a sparse relationship between the transforms.

In section 6 we derived a module for a given pair of b.c. by fixing (1) a base sequence of Chebyshev polynomials P_n depending on the left b.c. and (2), depending on the right b.c., a polynomial p in $\mathbb{C}[x]/p$. Since the recursion formula (4.3) for Chebyshev polynomials is symmetric, this can be done in a reverse way. We illustrate this with the pair DCT-3 and DST-3. The DST-3 has the left b.c. $a_{-1} = 0$ that fixes the base sequence U_ℓ, and it has the right b.c. $a_n = a_{n-2}$ that is fixed by $p = U_n - U_{n-2} = 0$. Alternatively, we can realize the same b.c. by the sequence T_ℓ, $\ell = -n+1, \ldots, 0$. Now the right b.c. are given by $a_{-1} = a_1$, i.e., $T_1 = T_{-1}$, which corresponds to $U_n - U_{n-2} = 0$. The left b.c. are fixed by $p = T_{-n} = T_n = 0$. The correspondence between the forward U_ℓ and the backward T_ℓ is as follows:

$$\begin{array}{l|ccc|l} 0 = U_{-1} & U_0, & \ldots, & U_{n-1} & U_n = U_{n-2}, \\ 0 = T_{-n} & T_{-(n-1)}, & \ldots, & T_0 & T_{-1} = T_1, \end{array}$$

where the vertical lines indicate the boundaries. In other words, using $T_{-\ell} = T_\ell$, the two bases $(U_0, \ldots, U_{n-1})$ and $(T_{n-1}, \ldots, T_0)$ afford identical representations of $A =$

$\mathbb{C}[x]/T_n$. Thus, the corresponding polynomial transforms must be scaled versions of each other. And indeed, if α_k denotes the zeros of T_n, we get, using basic trigonometric identities,

$$\begin{aligned} T_{n-1-\ell}(\alpha_k) &= \cos(n-1-\ell)\left(k+\tfrac{1}{2}\right)\pi/n \\ &= (-1)^k \cdot \sin\left(\ell+1\right)\left(k+\tfrac{1}{2}\right)\pi/n, \end{aligned}$$

and thus, using the definition of DST-3 and DCT-3 (Table 5.1),

$$\operatorname{diag}_{k=0}^{n-1}((-1)^k) \cdot \text{DST-3}_n = \text{DCT-3}_n \cdot \mathrm{J}_n, \tag{8.1}$$

where J_n denotes the opposite identity, i.e., the permutation matrix exchanging $i \leftrightarrow n-i$, $i = 0, \ldots, n-1$. Similar computations for all pairs of dual transforms yield the following result.

THEOREM 8.2 (translation by duality). *Let* DTT_n *and* DTT'_n *be a pair of dual transforms. Then*

$$\operatorname{diag}_{k=0}^{n-1}((-1)^k) \cdot \text{DTT}_n = \text{DTT}'_n \cdot \mathrm{J}_n\,.$$

In particular, dual DTTs have the same arithmetic complexity.

A second class of sparse relationships can be obtained in certain cases by appropriate base changes and will be explained in the following. Going back to Table 6.1, we see that the 16 DTTs are partitioned into four groups of four transforms each depending on the polynomial Q_n, which is essentially equal to one of the Chebyshev polynomials T_n, U_n, V_n, W_n. For example, on the main diagonal in Table 6.1 are all DTTs in the "U-group," which are exactly the self-dual transforms. Each of the other groups consists of two pairs of dual DTTs, respectively. For every two DTTs within the same group the corresponding algebra $\mathbb{C}[x]/Q_n$ is basically equal. The difference is in the basis chosen in the module. Thus, it is possible to derive a sparse relationship by performing an appropriate base change. We will illustrate this in the following two examples.

Example 8.3 (DCT-3 and DST-3). We consider again the pair DCT-3_n and DST-3_n. Using Table 6.1, we see that both transforms correspond to the same algebra, but with different bases,

$$\begin{aligned} \text{DCT-3}_n &\leftrightarrow \mathbb{C}[x]/T_n,\ b = (T_0, \ldots, T_{n-1}), \\ \text{DST-3}_n &\leftrightarrow \mathbb{C}[x]/T_n,\ b' = (U_0, \ldots, U_{n-1}), \end{aligned}$$

and that $\text{DCT-3}_n = [T_\ell(\alpha_k)]$ and $\text{DST-3}_n = D \cdot [U_\ell(\alpha_k)]$, where α_k are the zeros of T_n and $D = \operatorname{diag}_{k=0}^{n-1}(\sin(k+\frac{1}{2})\pi/n)$ arises from the scaling function. To compute the base change matrix B for $b \to b'$, we use that $T_\ell = \frac{1}{2}(U_\ell - U_{\ell-2})$ (by the 2nd row, 1st column in Table 4.3) and get

$$B = \frac{1}{2} \cdot \begin{bmatrix} 2 & 0 & -1 & & & \\ & 1 & 0 & -1 & & \\ & & \cdot & \cdot & \cdot & \\ & & & 1 & 0 & -1 \\ & & & & 1 & 0 \\ & & & & & 1 \end{bmatrix}.$$

Thus, $[T_\ell(\alpha_k)] = [U_\ell(\alpha_k)] \cdot B$, and hence

$$D \cdot \text{DCT-3}_n = \text{DST-3}_n \cdot B.$$

Note that this relationship is different from the one arising from the duality of DCT-3_n and DST-3_n (Theorem 8.2).

Example 8.4 (DCT-1 and DST-2). We will translate a DCT-1_{n+1} into a DST-2_n. Using Table 6.1 again, we get as associated algebras and bases

$$\begin{aligned} \text{DCT-1}_{n+1} &\leftrightarrow \mathbb{C}[x]/(x^2-1)U_{n-1}, \quad b &= (T_0,\dots,T_n),\\ \text{DST-2}_n &\leftrightarrow \mathbb{C}[x]/(x+1)U_{n-1}, \quad b' &= (W_0,\dots,W_{n-1}). \end{aligned}$$

Note that we have to choose size $n+1$ and n, respectively, to obtain comparable algebras. We have DCT-$1_{n+1} = [T_\ell(\alpha_k)]$ and DST-$2_n = D \cdot [W_\ell(\alpha_k)]$, where $\alpha_k = \cos k\pi/n$, $k = 0,\dots,n$, are the zeros of $(x^2-1)U_{n-1}$ (2nd row in Table 4.2). For the DST-2_n, $\alpha_0 = 1$ is skipped. The scaling matrix is $D = \operatorname{diag}_{k=0}^{n-1}(\sin(k+1)\pi/2n)$ (Table 6.2). We compute the base change matrix B for

$$\mathbb{C}[x]/(x^2-1)U_{n-1} \cong \mathbb{C}[x]/(x-1) \oplus \mathbb{C}[x]/(x+1)U_{n-1}.$$

The bases are $b, (1), b'$, respectively. Using $T_\ell = \frac{1}{2}(W_\ell - W_{\ell-1})$ (4th row, 3rd column in Table 4.3) and $T_n = \frac{1}{2}(W_n - W_{n-1}) \equiv -W_{n-1} \bmod (x+1)U_{n-1}$ (because, again from Table 4.3, $(x+1)U_{n-1} = \frac{1}{2}(W_n + W_{n-1})$), we get

$$B = \frac{1}{2} \cdot \begin{bmatrix} 2 & 2 & 2 & \cdot & \cdot & 2 \\ 2 & -1 & & & & \\ & 1 & -1 & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & -1 & \\ & & & & 1 & -2 \end{bmatrix}.$$

The 1's in the first row are due to $T_\ell(1) = 1$ (Lemma 4.2). We get $[T_\ell(\alpha_k)] = (\mathrm{I}_1 \oplus [W_\ell(\alpha_k)]) \cdot B$ and hence

$$(\mathrm{I}_1 \oplus D) \cdot \text{DCT-1}_{n+1} = (\mathrm{I}_1 \oplus \text{DST-2}_n) \cdot B.$$

We obtain the following theorem.

THEOREM 8.5 (translation by base change). *All DTTs of types* 1–4 *are in sparse relationship. All DTTs of types* 5–8 *are in sparse relationship.*

Proof. Similar computations as in Examples 8.3 and 8.4 show that all DTTs of types 1 and 2 (the "U-group") are in sparse relationship, and that all DTTs of types 3 and 4 (the "T-group") are in sparse relationship. By transposition, we obtain sparse relationship for DTTs of types 2 and 4 and thus for all DTTs of types 1–4, which is the first assertion. The other statement is proved analogously. ∎

Of particular importance is the translation between a DCT-4 and DCT-2, which, together with Theorem 8.6, yields a fast algorithm for the DCT-2 (see [28]).

Remarks. (1) Aside from Definition 8.1 there is another, more obvious, form of duality among the DTTs: DTT and DTT′ are dual if $\text{DTT}^T = \text{DTT}'$. Currently, we have no algebraic explanation for this duality. (2) Note that "sparse relationship" does not define an equivalence relation. Every two matrices (of the same size) can be converted into each other using a (long enough) sequence of sparse matrices.

8.2. Direct sum: Fast algorithms via polynomial factorization. In this section we will derive recursive algorithms for all DTTs in the U-group, i.e., the DCT and DST of types 1 and 2. The algorithms are based on the rational factorization of the polynomials U_n given in Lemma 4.2(ii)–(iii).

As an example we will consider a DCT-2_n, where $n = 2m$. Consulting Table 6.1, we get as corresponding algebra $\mathbb{C}[x]/(x-1)U_{n-1}$ with basis $b = (V_0, \ldots, V_{n-1})$. Lemma 4.2(ii) gives the factorization $U_{2m-1} = 2 \cdot U_{m-1} \cdot T_m$, which leads to the isomorphism

$$\mathbb{C}[x]/(x-1)U_{2m-1} \cong \mathbb{C}[x]/(x-1)U_{m-1} \oplus \mathbb{C}[x]/T_m. \tag{8.2}$$

For the summands we choose bases b, b', b', respectively, $b' = (V_0, \ldots, V_{m-1})$. The zeros of $(x-1)U_{n-1}$ are $\cos k\pi/n$, $k = 0, \ldots, n-1$. Thus, the first summand in (8.2) collects the zeros with even k, and the second summand the zeros with odd k (cf. Table 4.2). Now the decomposition of DCT-2_n follows Lemma 7.1. To compute the base change matrix B in (8.2) we need

$$\begin{aligned} &V_{m+k} \equiv V_{m-k-1} \bmod (x-1)U_{m-1} \text{ and} \\ &V_{m+k} \equiv -V_{m-k-1} \bmod T_m, \end{aligned}$$

which can be shown by induction using $(x-1)U_{m-1} = V_m - V_{m-1}$, $T_m = V_m + V_{m+1}$, and (4.3). We get

$$B = \begin{bmatrix} 1 & & & & & 1 \\ & . & & & . & \\ & & 1 & 1 & & \\ 1 & & & & & -1 \\ & . & & & . & \\ & & 1 & -1 & & \end{bmatrix} = \begin{bmatrix} \mathrm{I}_m & \mathrm{J}_m \\ \mathrm{I}_m & -\mathrm{J}_m \end{bmatrix} = (\mathrm{DFT}_2 \otimes \mathrm{I}_m)(\mathrm{I}_m \oplus \mathrm{J}_m).$$

The summands in (8.2) are decomposed recursively using a DCT-2_m and DCT-4_m, respectively. The resulting one-dimensional summands are permuted in canonical order using the stride permutation L^n_m (see section 2). Since DCT-2 and DCT-4 have the same scaling function, we get

$$\text{DCT-2}_{2m} = \mathrm{L}^{2m}_m \cdot (\text{DCT-2}_m \oplus \text{DCT-4}_m) \cdot B.$$

Besides DCT-2, similar derivations can be performed on the three remaining transforms in the U-group, DCT-1, DST-1, and DST-2, using $U_{2m-1} = 2U_{m-1}T_m$ and using $U_{2m} = V_m W_m$. The complete set of identities can be stated using two types of block matrices and two types of permutation matrices. The block matrices give the base change

$$B_{2m} = \begin{bmatrix} \mathrm{I}_m & \mathrm{J}_m \\ \mathrm{I}_m & -\mathrm{J}_m \end{bmatrix}, \quad B_{2m+1} = \begin{bmatrix} \mathrm{I}_m & 0 & \mathrm{J}_m \\ 0 & 1 & 0 \\ \mathrm{I}_m & 0 & -\mathrm{J}_m \end{bmatrix},$$

and the permutation matrices give the reordering of the irreducible modules

$$\begin{aligned} &P_{2m} = \mathrm{L}^{2m}_m, \\ &P_{2m+1} : \ i \to (m+1)i \bmod 2m+1, \quad i = 0, \ldots, 2m. \end{aligned}$$

THEOREM 8.6. *The following recursive algorithms for DTTs are based on the rational factorization $U_{2m-1} = 2 \cdot U_{m-1} \cdot T_m$. We also indicate where they first appeared in the literature (to our best knowledge).*

(i) DCT-$1_{2m+1} = P_{2m+1} \cdot (\text{DCT-1}_{m+1} \oplus \text{DCT-3}_m) \cdot B_{2m+1}$ [27].

(ii) $\text{DST-1}_{2m-1} = P_{2m-1} \cdot (\text{DST-3}_m \oplus \text{DST-1}_{m-1}) \cdot B_{2m-1}$ [55].
(iii) $\text{DCT-2}_{2m} = P_{2m} \cdot (\text{DCT-2}_m \oplus \text{DCT-4}_m) \cdot B_{2m}$ [7].
(iv) $\text{DST-2}_{2m} = P_{2m} \cdot (\text{DST-4}_m \oplus \text{DST-2}_m) \cdot B_{2m}$ [52].

Theorem 8.6 is complemented by the decompositions in the following theorem. We did not find these in the literature.

THEOREM 8.7. *The following recursive algorithms for DTTs are based on the rational factorization* $U_{2m} = V_m W_m$.

(i) $\text{DCT-1}_{2m} = P_{2m} \cdot (\text{DCT-5}_m \oplus \text{DCT-7}_m) \cdot B_{2m}$.
(ii) $\text{DST-1}_{2m} = P_{2m} \cdot (\text{DST-7}_m \oplus \text{DST-5}_m) \cdot B_{2m}$.
(iii) $\text{DCT-2}_{2m+1} = P_{2m+1} \cdot (\text{DCT-6}_{m+1} \oplus \text{DCT-8}_m) \cdot B_{2m+1}$.
(iv) $\text{DST-2}_{2m+1} = P_{2m+1} \cdot (\text{DST-8}_{m+1} \oplus \text{DST-6}_m) \cdot B_{2m+1}$.

Remarks. (1) Transposition of the decompositions in Theorems 8.6 and 8.7 yields algorithms for the DTTs of type 3. (2) Theorem 8.6 reveals why the DCT-1 and the DST-1 are usually considered at sizes $2^k + 1$ and $2^k - 1$, respectively. The available algorithms are more efficient since there are no simple sparse factorizations of the DTTs of types 5–8. (3) Combining Theorems 8.5 and 8.6 gives a complete set of algorithms for the DTTs of types 1–4, of 2-power size. (For type 1 the size differs by 1; see remark (2).) (4) It is possible to derive algorithms for the more general case $n = k\ell$, using the factorization in Lemma 4.2(ii).

8.3. Reduction: Fast algorithms via polynomial decomposition. In this section we derive algorithms based on the decomposition of the polynomial T_n in Lemma 4.2(i). This decomposition property allows the decomposition of all *DTTs* in the T-group, i.e, DCT and DST of types 3 and 4, using Theorem 7.3.

As an example we will consider a DCT-3_n, where $n = 2m$. Using Table 6.1, we get the corresponding algebra $\mathbb{C}[x]/T_n$ with basis $b = (T_0, \ldots, T_{n-1})$. We use the decomposition $T_{2m} = T_m(T_2)$ (Lemma 4.2). Following Theorem 7.3 and its proof, we choose bases $c = (T_0, \ldots, T_{m-1})$ and $d = (T_0, T_1)$ of $\mathbb{C}[x]/T_m$ and $\mathbb{C}[x]/T_2$, respectively. We get the new basis

$$b' = (T_0, T_2, \ldots, T_{2m-2}, T_1, (T_1 + T_3)/2, \ldots, (T_{2m-3} + T_{2m-1})/2).$$

Thus, the base change $b' \to b$ is given by

$$B = \begin{bmatrix} 1 & & & 0 & & & \\ 0 & 0 & & 1 & \frac{1}{2} & & \\ 0 & 1 & & 0 & 0 & & \\ 0 & 0 & & 0 & \frac{1}{2} & \frac{1}{2} & \\ & \vdots & & & & \vdots & \\ & & & & & & \frac{1}{2} \\ & & 1 & & & & 0 \\ & & 0 & & & & \frac{1}{2} \end{bmatrix},$$

and the base change $b \to b'$ by B^{-1}. The zeros of T_n are $\alpha_k = \cos(k + \frac{1}{2})\pi/n$, $\alpha_k = -\alpha_{n-1-k}$, and $T_2(\alpha_k) = T_2(\alpha_{n-1-k})$. Thus, the permutation P in Theorem 7.3 is $P = (\text{I}_m \oplus \text{J}_m)$. Further,

$$\mathcal{P}_{d,\overline{\alpha}_i} = \begin{bmatrix} T_0(\alpha_i) & T_1(\alpha_i) \\ T_0(\alpha_{n-1-i}) & T_1(\alpha_{n-1-i}) \end{bmatrix} = \begin{bmatrix} 1 & \alpha_i \\ 1 & -\alpha_i \end{bmatrix} = \text{DFT}_2 \cdot \text{diag}(1, \alpha_i)$$

for $i = 0, \ldots, m$. This can be used to derive $(\bigoplus_{i=0}^{m-1} \mathcal{P}_{d,\overline{\alpha}_i})^{\text{L}_m^{2m}} = (\text{DFT}_2 \otimes \text{I}_m) \cdot$

$(\mathrm{I}_m \oplus \mathrm{diag}_{i=0}^{m-1}(\alpha_i))$, and we get

$$\text{DCT-3}_{2m} = P \cdot (\mathrm{DFT}_2 \otimes \mathrm{I}_m) \cdot (\mathrm{I}_m \oplus \mathrm{diag}_{i=0}^{m-1}(\alpha_i)) \cdot (\mathrm{I}_2 \otimes \text{DCT-3}_m) \cdot B^{-1}.$$

Further simplification can be achieved by writing $B = C \cdot (\mathrm{I}_m \oplus \frac{1}{2}\,\mathrm{I}_m)$ and observing that $\mathrm{I}_m \oplus 2\,\mathrm{I}_m$ commutes with $\mathrm{I}_2 \otimes \text{DCT-3}_m$. For simplicity we set $D = \mathrm{diag}_{i=0}^{m-1}(\alpha_i)$ and get

$$\text{DCT-3}_{2m} = P \cdot (\mathrm{DFT}_2 \otimes \mathrm{I}_m) \cdot (\mathrm{I}_m \oplus 2D) \cdot (\mathrm{I}_2 \otimes \text{DCT-3}_m) \cdot C^{-1}. \tag{8.3}$$

Equation (8.3) is also a good example for studying the effect of transposition and inversion on deriving new algorithms from known ones. Transposition of (8.3) is straightforward and yields

$$\text{DCT-2}_{2m} = C^{-T} \cdot (\mathrm{I}_2 \otimes \text{DCT-2}_m) \cdot (\mathrm{I}_m \oplus 2D) \cdot (\mathrm{DFT}_2 \otimes \mathrm{I}_m) \cdot P. \tag{8.4}$$

For the inversion of (8.3) we need $\text{DCT-3}_n^{-1} = \frac{2}{n} \cdot \mathrm{diag}(\frac{1}{2}, 1, \ldots, 1) \cdot \text{DCT-2}_n$. After simplifications we get

$$\text{DCT-2}_{2m} = C_1 \cdot (\mathrm{I}_2 \otimes \text{DCT-2}_m) \cdot (\mathrm{I}_m \oplus (2D)^{-1}) \cdot (\mathrm{DFT}_2 \otimes \mathrm{I}_m) \cdot P, \tag{8.5}$$

where C_1 arises from C by setting the entry 2 at position $(2, m+1)$ to 1. C_1 incurs only additions. Transposing (8.5) (or, equivalently, inverting (8.4)) yields again an algorithm for DCT-3,

$$\text{DCT-3}_{2m} = P \cdot (\mathrm{DFT}_2 \otimes \mathrm{I}_m) \cdot (\mathrm{I}_m \oplus (2D)^{-1}) \cdot (\mathrm{I}_2 \otimes \text{DCT-3}_m) \cdot C_1^T. \tag{8.6}$$

Equations (8.5) and (8.6) are very similar to (8.4) and (8.3), respectively, with the difference that inverting the entries in the diagonal (middle factor) saves one multiplication by 2 in the base change matrix (C_1 vs. C). More crucial, the additions in C_1 and C_1^T can be performed in parallel (i.e., the critical path has length 1), which does not hold for C^{-1} and C^{-T}.

Each of the equations (8.3)–(8.6) occurs in the literature. The references are (8.3) and (8.4) in [24], (8.5) in [57] (transposed definition of DTTs), and (8.6) in [28] and [56] (transposed definition of DTTs).

Note that (8.4) can also be obtained by first applying Theorem 8.6(iii) and then translating the resulting DCT-4 using Theorem 8.5.

Similar computations for the other DTTs in the T-group yield the following result.

THEOREM 8.8. *Let $n = 2m$. All DTTs in the T-group have a fast recursive algorithm of the form*

$$\mathrm{DTT}_{2m} = P \cdot (\mathrm{DFT}_2 \otimes \mathrm{I}_m) \cdot (\mathrm{I}_m \oplus D) \cdot (\mathrm{I}_2 \otimes \mathrm{DTT}_m) \cdot B,$$

where P is a permutation matrix, D is diagonal, and B is sparse. This factorization is based on $T_{2m} = T_m(T_2)$, and the concrete form of P and B can be obtained using Theorem 7.3.

For the DST-3 the factorization can also be found in [56]. For DCT-4 and DST-4, the factorizations do not appear in the literature. They are less efficient with respect to arithmetic cost.

Remark. It is possible to derive a recursive algorithm based on $T_{k\ell} = T_n(T_m)$ using Theorem 7.3. The problem for larger m is the further decomposition of the occurring matrices $\mathcal{P}_{d,\overline{\alpha}_i}$ in Theorem 7.3.

9. Fast DTTs via group symmetries. In this section we will derive fast DTT algorithms that are based on "group symmetries" in the sense defined below. In the cases where they occur, these symmetries are a direct consequence of the DTT properties in Theorem 6.2. We will identify two ways in which group symmetries might come into play.

1. *Extension* (section 9.2). By extension to a group algebra of the algebra $A = \mathbb{C}[x]/p$ associated to a DTT.

2. *Automorphisms* (section 9.3). By subgroups of the automorphism group of A.

These symmetries lead to algorithms that are substantially different from the ones derived in section 8.

For the convenience of the reader, we briefly overview group symmetry–based matrix factorization. In the following we take a "representation" approach, instead of the equivalent "module-with-basis" point of view.

9.1. Group symmetry–based matrix factorization. Matrix factorization based on group symmetries has its origin in [31, 32] and was generalized in [15, 36, 37, 19] to the form presented here. In [18] the technique was successfully applied to several discrete signal transforms, which initiated the research presented in this paper. Due to space limitations we can give only a brief overview and refer to those references for further details.

In the following, G is a finite solvable group. All representations of G (or, equivalently, of $\mathbb{C}[G]$) in the following arise from *right* G-modules. The entire approach is based on the following definition of symmetry.

DEFINITION 9.1. *Let B be an arbitrary complex matrix. A pair (ϕ_1, ϕ_2) of representations of G is called a* symmetry *of B if*

$$\phi_1(g) \cdot B = B \cdot \phi_2(g) \quad \textit{for } g \in G.$$

G is then called a symmetry group *of B.*

If B has a symmetry, we can factor B according to Figure 9.1. We choose matrices A_1, A_2 that decompose ϕ_1, ϕ_2, respectively, into a direct sum of irreducible representations ρ_1 and ρ_2. Then we compute the matrix

$$D = A_1^{-1} \cdot B \cdot A_2$$

so that the diagram commutes. We obtain the factorization

$$B = A_1 \cdot D \cdot A_2^{-1}.$$

The matrix D is sparse since it is the conjugating matrix for two reduced representations ρ_1, ρ_2 (a consequence of Schur's lemma [12]). This means that the factorization of B is useful as a fast algorithm for B if the A_i's are sparse or can themselves be written as products of sparse matrices. This is possible in at least the following two cases: (1) ϕ_i is a permuted direct sum of irreducible representations, i.e., $\phi_i = \rho_i^P$, where P is a permutation matrix. In this case we say that ϕ_i is of type "irred." It is $A_i = P^{-1}$. (2) ϕ_i is monomial. (A representation is monomial if all its images are monomial matrices, i.e., have exactly one nonzero entry in each row and column.) In this case we say that ϕ_i is of type "mon." The decomposition matrix A_i can be determined as a product of sparse matrices using the algorithm in [37]. Briefly sketched, this algorithm translates the monomial representation into

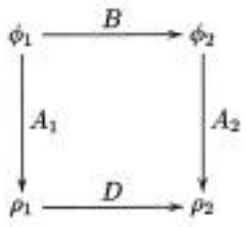


FIG. 9.1. *Factorization of the matrix M with symmetry (ϕ_1, ϕ_2).*

TABLE 9.1
Types of symmetry that can be used for factorizing B (irrs = irreducible representations).

mon-mon symmetry	ϕ_1 and ϕ_2 monomial
mon-irred symmetry	ϕ_1 monomial, ϕ_2 permuted direct sum of irrs
irred-mon symmetry	ϕ_2 monomial, ϕ_1 permuted direct sum of irrs

an induction that is decomposed stepwise along a composition series using certain recursion formulas similar to Theorem 7.3. We say that ϕ_i is of type "mon."

Depending on the types of the ϕ_i, we obtain the three types of symmetry shown in Table 9.1. We have omitted the type "irred-irred" since it requires that B already be sparse.

Algorithms for finding symmetry [19] and the algorithm [37] for the stepwise decomposition of monomial representations have been implemented as part of the GAP [22] share package AREP [17, 16] for constructive group representation theory. Thus, AREP can find these factorizations automatically and can be used as a discovery tool for sparse matrix factorizations, i.e., fast algorithms.

In the remainder of this section, we will show that mon-irred symmetries as well as mon-mon symmetries occur among the DTTs and how these symmetries can be derived. We will also discuss the structure of the resulting algorithms.

9.2. Algorithms by extension to group algebras. In this section we will show which DTTs possess a mon-irred symmetry that can be used for deriving fast algorithms. In [18] exactly four DTTs exhibited a mon-irred symmetry with dihedral symmetry groups in all cases. The transforms were the DCT and DST of types 3 and 4. We will now explain and derive these symmetries. Note that we will deal with right representations (arising from right modules) to comply with the symmetry definition 9.1. A right representation is the transpose of a left representation.

We start with the general case of a scaled polynomial transform. As usual, let b be a basis of $A = \mathbb{C}[x]/p$, and let α be the zero vector of p. Further, let f be a scaling function. If ϕ is the *right* representation afforded by the regular module A (or, equivalently, $f \cdot A$), then, by Lemma 3.6,

$$\phi \cdot \mathcal{P}^T_{f \cdot b, \alpha} = \mathcal{P}^T_{f \cdot b, \alpha} \cdot \rho,$$

where ρ is a direct sum of one-dimensional irreducible representations of A. If ϕ can be extended to a representation $\overline{\phi}$ of a group algebra $\mathbb{C}[G]$ of a finite group G, then ρ extends to a permuted direct sum of irreducible representations of $\mathbb{C}[G]$. (On extension, the one-dimensional irreducibles in ρ—not necessarily adjacent ones—may

fuse to irreducibles of $\mathbb{C}[G]$ of larger dimension.) In other words, $\mathcal{P}^T_{f\cdot b,\alpha}$ decomposes $\overline{\phi}$, up to a permutation. We obtain the following result.

LEMMA 9.2. *We use previous notation. If the right regular representation ϕ of $A = \mathbb{C}[x]/p$ can be extended to a representation $\overline{\phi}$ of a group algebra $\mathbb{C}[G]$, where G is finite, then*

$$\overline{\phi} \cdot \mathcal{P}^T_{f\cdot b,\alpha} = \mathcal{P}^T_{f\cdot b,\alpha} \cdot \overline{\rho},$$

where $\overline{\rho}$ is a permuted direct sum of irreducible representations of $\mathbb{C}[G]$. If, in particular, $\overline{\phi}$ is monomial, then $\mathcal{P}^T_{f\cdot b,\alpha}$ has a mon-irred symmetry, and $\mathcal{P}_{f\cdot b,\alpha}$ has an irred-mon symmetry, both with symmetry group G.

Now we will apply Lemma 9.2 to determine which DTTs possess a mon-irred symmetry. Consider a fixed DTT with associated regular representation ϕ. The representation ϕ can be extended to a monomial representation iff all images $\phi(q)$, $q \in A$, can be written as a linear combination of monomial matrices. Since A is cyclic, it is sufficient to consider the images of the generator $\phi(x)$, which is given by the corresponding matrix $B(\cdot)$ in (5.1).

THEOREM 9.3. *The four transforms* DCT_n *and* DST_n *of types* 3 *and* 4, $n \geq 0$, *are the only DTTs that have a mon-irred symmetry (ϕ, ρ). Denote by $D_{2k} = \langle \sigma, \tau \mid \sigma^2 = \tau^2 = (\sigma\tau)^k = 1 \rangle$ the dihedral group with $2k$ elements. Further, let, for even n,*

$$\pi_1 = (1,2)(3,4), \ldots, (n-1,n) \quad \textit{and} \quad \pi_2 = (2,3)(4,5), \ldots, (n-2,n-1),$$

and, for odd n,

$$\pi_1 = (1,2)(3,4), \ldots, (n-2,n-1) \quad \textit{and} \quad \pi_2 = (2,3)(4,5), \ldots, (n-1,n)$$

(viewed as a permutation on $\{1, \ldots, n\}$). The symmetry group for DCT-3_n *and* DST-3_n *is D_{2n}, for* DCT-4_n *and* DST-4_n *is D_{4n}. The respective monomial representation ϕ is given for even n by*

$$\begin{array}{ll}
\text{DCT-3}_n: & \sigma \mapsto [\pi_1, n],\ \tau \mapsto [\pi_2, n], \\
\text{DST-3}_n: & \sigma \mapsto [\pi_1, n],\ \tau \mapsto [\pi_2, (-1, 1, \ldots, 1, -1)], \\
\text{DCT-4}_n: & \sigma \mapsto [\pi_1, n],\ \tau \mapsto [\pi_2, (1, \ldots, 1, -1)], \\
\text{DST-4}_n: & \sigma \mapsto [\pi_1, n],\ \tau \mapsto [\pi_2, (-1, 1 \ldots, 1,)],
\end{array}$$

and for odd n by

$$\begin{array}{ll}
\text{DCT-3}_n: & \sigma \mapsto [\pi_1, n],\ \tau \mapsto [\pi_2, n], \\
\text{DST-3}_n: & \sigma \mapsto [\pi_1, (1, \ldots, 1, -1)],\ \tau \mapsto [\pi_2, (-1, 1, \ldots, 1)], \\
\text{DCT-4}_n: & \sigma \mapsto [\pi_1, (1, \ldots, 1, -1)],\ \tau \mapsto [\pi_2, n], \\
\text{DST-4}_n: & \sigma \mapsto [\pi_1, n],\ \tau \mapsto [\pi_2, (-1, 1 \ldots, 1,)].
\end{array}$$

Proof. For all 16 DTTs and their associated representations ϕ, we have to consider the matrices $\phi(x) = B(\beta_1, \beta_2, \beta_3, \beta_4)$, with β_i as given in Table 5.2. Because of its structure, $B(\cdot)$ can be written as a linear combination of monomial matrices iff it can be written as the sum of two monomial matrices. Assume $\beta_1 = 0$. Writing $B(0, \ldots)$ as the sum of two monomial matrices M_1, M_2 requires that both M_1 and M_2 have an entry $\neq 0$ at position $(1,2)$. Since the entry $(3,2)$ of $B(0, \ldots)$ is also $\neq 0$, this decomposition is not possible. Analogously, a decomposition is not possible if $\beta_4 = 0$. In the remaining four cases the decomposition is possible and yields the desired results. We will give one case as an example. It is readily verified that

$$B(1,1,1,1) = [\pi_1, n] + [\pi_2, n].$$

The permutations π_1, π_2 are involutions and hence generate a dihedral group D_{2m}. The number m is the order of the product $\pi_1\pi_2$, here n. By Theorem 6.2 and Table 5.2, $B(1,1,1,1)$ is diagonalized by DCT-3_n, which proves the result. The other three cases can be treated analogously. □

Remark. Theorem 9.3 explains the symmetries found in [18].

We want to briefly sketch the decomposition procedure for a DCT-4. For full details we refer the reader to [18, 19, 37].

Example 9.4 (DCT-4). We consider a DCT-4 of size $n = 2^k$. By Theorem 9.3, the matrix $B = \text{DCT-4}_{2^k}$ has a mon-irred symmetry (ϕ, ρ) with dihedral symmetry group $D_{2^{k+2}}$. We follow Figure 9.1. The decomposition algorithm will decompose ϕ stepwise along the composition series

$$D_{2^{k+2}} \geq D_{2^{k+1}} \geq \cdots \geq D_{2^2},$$

using a recursion formula for the induction of representations. Note that the last representation of D_{2^2} is decomposed since it is of dimension 1. This gives rise to a factorized decomposition matrix A_1 of ϕ. The representation ρ is a permuted direct sum of irreducible representations and can thus be decomposed by a permutation matrix A_2. The correction matrix D is computed as $D = A_1^{-1} \cdot B \cdot A_2$ to yield

$$\text{DCT-4}_{2^k} = A_1 \cdot D \cdot A_2^{-1}.$$

As an example, we give a factorization of a DCT-4_8 as it is automatically found by AREP,

$$\begin{aligned}
\text{DCT-4}_8 \quad = \quad & [(1,2,8)(3,6,5), (1,-1,1,1,1,-1,1,1)] \\
& \cdot (\mathrm{I}_2 \otimes ((\mathrm{I}_2 \oplus \tfrac{1}{\sqrt{2}} \cdot \mathrm{DFT}_2) \cdot [(3,4),4] \cdot (\mathrm{DFT}_2 \otimes \mathrm{I}_2))) \\
& \cdot [(1,3)(2,4)(5,7)(6,8),8] \cdot (\mathrm{I}_4 \oplus \mathrm{R}_{\frac{15}{8}\pi} \oplus \mathrm{R}_{\frac{11}{8}\pi}) \\
& \cdot (\mathrm{DFT}_2 \otimes \mathrm{I}_4) \cdot [(3,5,7)(4,6,8),8] \\
& \cdot \tfrac{1}{2} \cdot (\mathrm{R}_{\frac{31}{32}\pi} \oplus \mathrm{R}_{\frac{19}{32}\pi} \oplus \mathrm{R}_{\frac{27}{32}\pi} \oplus \mathrm{R}_{\frac{23}{32}\pi}) \\
& \cdot [(1,8,5,6,3,2)(4,7),8].
\end{aligned} \tag{9.1}$$

The (factorized) matrix A_1 is given in lines 1–4, the matrix D in line 5, and the matrix A_2^{-1} in line 6 (the last line).

We observe that the factorization in (9.1) contains rotation matrices

$$R_a = \begin{bmatrix} \cos(a) & \sin(a) \\ -\sin(a) & \cos(a) \end{bmatrix},$$

which do not occur in the algorithms derived in section 8. The general (arbitrary $n = 2^k$) version of this algorithm can be found in [7] (corrected in [51, 52]). Combining this algorithm with Theorem 8.6(iii) yields a factorization of DCT-2, and thus, by transposition, of DCT-3, into rotation matrices [7]. The obtained algorithm coincides with the one derived from the mon-irred symmetry of the DCT-3.

Note that the algorithms arising from a mon-irred symmetry occur only in an iterative form in the literature; i.e., the transform matrix is completely factorized (as in (9.1)) and not into transforms of smaller size. The reason is in the decomposition procedure (cf. Figure 9.1), since not B, but A_1 is decomposed recursively.

Remark. It is striking that, e.g., the algorithm for a DCT-3_{2^k} arising from its mon-irred symmetry and the algorithm from Theorem 8.8 have precisely the same arithmetic cost [7, 28, 56].

9.3. Algorithms from automorphism groups. In section 9.2 we showed how, in certain cases, a mon-irred symmetry of a DTT can be derived from its interpretation as a (scaled) polynomial transform. In the following we will show that a—completely different—type of mon-mon symmetry also occurs among the DTTs. This type of symmetry, if present, arises from the automorphism group of the associated algebra. All modules in this section will be right modules.

We introduce the following notation. Let $A = \mathbb{C}[x]/p$. Automorphisms of A will be denoted by letters g, h. We multiply automorphisms from left to right; i.e., in gh, g is applied before h. This complies with applying automorphisms from the right; i.e., if $q \in A$, we write q^g for the image of q under g. If ϕ is a representation of A, and g an automorphism, then $\phi^g : \ q \mapsto \phi(q^g)$ defines another representation of A. As suggested by this notation, $(\phi^g)^h = \phi^{gh}$.

A possible source of a mon-mon symmetry of a polynomial transform $\mathcal{P}_{b,\alpha}$ is described in the following theorem.

THEOREM 9.5. *Let $A = \mathbb{C}[x]/p$ be a regular module with basis b. The polynomial p is separable and has zeros $\alpha = (\alpha_0, \ldots, \alpha_{n-1})$. Denote by ϕ the (right regular) representation afforded by A and b. Assume that A has a group G of automorphisms with the property that for each $g \in G$ there exists a monomial matrix M_g with*

$$\phi^g = \phi^{M_g{}^{-1}}. \tag{9.2}$$

Then $\mathcal{P}_{b,\alpha}^T$ has a mon-mon symmetry (χ, ψ) with symmetry group $\overline{G} \cong \langle M_g \mid g \in G \rangle$. Then $G \cong \overline{G}/N$, where $N \trianglelefteq \overline{G}$ denotes the normal subgroup defined by

$$\begin{aligned} g' \in N &\Leftrightarrow \phi(q) \cdot \chi(g') = \chi(g') \cdot \phi(q) \quad \textit{for all } q \in A \\ &\Leftrightarrow \chi(g') \in \phi(A). \end{aligned}$$

If D is any invertible diagonal matrix, then $(D \cdot \mathcal{P}_{b,\alpha})^T = \mathcal{P}_{b,\alpha}^T \cdot D$ has the same mon-mon symmetry as $\mathcal{P}_{b,\alpha}^T$.

Proof. First we note that the set $S = \{M_g \mid g \in G\}$ is not a group, since for every g there are (if any) many possible choices for M_g, e.g., all $a \cdot M_g$, where $a \in \mathbb{C}$. Conversely, every $M_g \in S$ uniquely defines an automorphism of A, since $\phi^g = \phi^h$, and ϕ faithful, implies $g = h$. Now we reverse the situation by defining a mapping $\gamma : \ S \to G$, $M_g \mapsto g$. Let $\overline{G} = \langle S \rangle$ (the group generated by S). Then γ can be extended to a homomorphism $\overline{\gamma} : \ \overline{G} \to G$, since, for $M, M' \in S$ and using (9.2),

$$\phi^{\overline{\gamma}(MM')} = \phi^{(MM')^{-1}} = (\phi^{M'^{-1}})^{M^{-1}} = (\phi^{\overline{\gamma}(M')})^{M^{-1}} = (\phi^{M^{-1}})^{\overline{\gamma}(M')} = \phi^{\overline{\gamma}(M)\overline{\gamma}(M')}.$$

By definition, $\overline{\gamma}$ is surjective, and the kernel of $\overline{\gamma}$ is given by $N = \{M \mid \phi = \phi^M\}$, and thus $G \cong \overline{G}/N$. Since $M \in N$ implies that M commutes with each $\phi(q)$, $q \in A$, $M \in \phi(A)$. Viewing $\overline{G}$ as a monomial representation χ of itself shows all assertions on $\overline{G}$.

It remains to show that $\mathcal{P}_{b,\alpha}^T$ has a mon-mon symmetry (χ, ψ). To this end we choose an arbitrary monomial matrix $M = \chi(M)$ in $\overline{G}$. The representation ϕ is decomposed by $\mathcal{P}_{b,\alpha}^T$ into a direct sum of irreducible representations ρ (cf. Lemma 3.2). Thus, $\phi^{\overline{\gamma}(M)}$ is also decomposed by $\mathcal{P}_{b,\alpha}^T$ into a direct sum of irreducible representations ρ'. Following Figure 9.2, there is a unique matrix M' such that

$$M \cdot \mathcal{P}_{b,\alpha}^T = \mathcal{P}_{b,\alpha}^T \cdot M'.$$

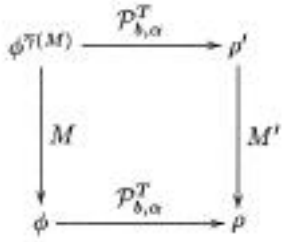


FIG. 9.2. *Constructing a mon-mon symmetry for* $\mathcal{P}^T_{b,\alpha}$.

Since M' conjugates ρ' onto ρ, it is monomial. Setting $\psi(M) = M'$ defines a monomial representation of $\overline{G}$ and shows that $\mathcal{P}^T_{b,\alpha}$ has the mon-mon symmetry (χ, ψ).

If D is any invertible diagonal matrix, then $\rho^D = \rho$ and ${\rho'}^D = \rho'$, since all irreducible summands of ρ, ρ' are of dimension 1. Thus, we can replace $\mathcal{P}^T_{b,\alpha}$ by $\mathcal{P}^T_{b,\alpha} \cdot D$ in Figure 9.2, obtaining the same mon-mon symmetry. This completes the proof. □

Remarks. (1) $\mathcal{P}^T_{b,\alpha}$ has the mon-mon symmetry (χ, ψ) iff $\mathcal{P}_{b,\alpha}$ has the mon-mon symmetry (ψ^T, χ^T). (2) The last assertion in Theorem 9.5 shows that we can apply it to scaled polynomial transforms and thus to the DTTs.

In the remainder of this section, we will use Theorem 9.5 to derive the mon-mon symmetries for the DTTs whose transposes are in the T-group, i.e., those with associated algebra $\mathbb{C}[x]/T_n$ (cf. Table 6.1) for the special case where $n = 2^m$ is a 2-power. A complete investigation of all DTTs and sizes would exceed the space available.

First we need a suitable group G of automorphisms of $A = \mathbb{C}[x]/T_n$.

LEMMA 9.6. *Let* $n = 2^m$ *and* $A = \mathbb{C}[x]/T_n$. *Each mapping*

$$g_k : T_1 \mapsto T_k \quad \text{and} \quad g_{-k} : T_1 \mapsto -T_k, \qquad 1 \le k \le n, \ k \text{ odd},$$

defines an automorphism of the algebra A. *The set* G_n *of all such* $g_{\pm k}$ *is a cyclic group of order* n.

Proof. Before we start the proof we investigate the sequence T_k, $k \ge 0$, in A. The following two equations allow the reduction of each T_k modulo T_n:

$$\begin{aligned} 0 \equiv T_n T_{n-k} = \tfrac{1}{2}(T_{2n-k} + T_k) \quad &\Rightarrow \quad T_k \equiv -T_{2n-k}, \\ 0 \equiv T_n T_{n+k} = \tfrac{1}{2}(T_{2n+k} + T_k) \quad &\Rightarrow \quad T_k \equiv -T_{2n+k}. \end{aligned} \tag{9.3}$$

The latter equation also shows that $T_k \equiv T_{k+4n}$, i.e., the sequence T_k, $k \ge 1$, has period $4n$ (in A). Using (9.3), we can compute the reduced T_k, $k = 0, \ldots, 4n-1$, as

$$T_0, \ldots, T_{n-1} \mid 0 \ -T_{n-1}, \ldots, -T_1 \mid -T_0, \ldots, -T_{n-1} \mid 0 \ T_{n-1}, \ldots, T_1 \mid, \tag{9.4}$$

where the vertical lines indicate the reflection points at multiples of n.

Now we start the proof of Lemma 9.6. Let $n = 2^m$. We will repeatedly use that T_n is an even function and that T_k, k odd, is an odd function. Also note that $g_{\pm k}$ maps $T_\ell = T_\ell(T_1) \mapsto T_\ell(\pm T_k)$.

(1) $g_{\pm k}$ is a homomorphism, since $T_n(\pm T_k) = T_n(T_k) = T_k(T_n) \equiv 0$ (Lemma 4.2, (i)); i.e., the defining equation $T_n = 0$ in A is preserved. (2) $g_{\pm k}$ is invertible; G_n is

a group. Let g_k be given, k odd. We choose an ℓ with $k\ell \equiv 1 \bmod 4n$. The mapping $T_1 \mapsto T_\ell$ inverts g_k, since $T_{k\ell} \equiv T_1$ (see beginning of this proof). Similarly, $T_1 \mapsto -T_\ell$ inverts g_{-k}. Using (9.3), we can reduce $T_\ell \equiv T_{\ell'}$ or $\equiv -T_{\ell'}$ for a suitable odd $\ell' < n$. This shows that G_n is closed under inversion. Also $g_{\pm k} g_{\pm \ell} : T_1 \mapsto \pm T_\ell(\pm T_k) = \mp T_{k\ell}$, which can be reduced analogously. Thus, G_n is a group. (3) G_n is cyclic. For $n = 2$, g_{-1} has order 2. For $n = 4$, g_3 has order 4 ($T_3(T_3) = T_9 \equiv -T_1$). For $n > 4$, we show that g_5 has order n. Observing (9.4), we get that g_5^e is the identity iff $5^e \equiv \pm 1 \bmod 4n$. Since 5^e is never $\equiv -1$ and 5 has order $n \bmod 4n$ ($n = 2^m$), we get the desired assumption. □

Now we will use the group G_n of automorphisms (Lemma 9.6) and Theorem 9.5 to derive mon-mon symmetries for all DTTs whose inverses are in the T-group.

THEOREM 9.7. *Let $n = 2^m \geq 4$ and G_n be as defined in Lemma 9.6. The transforms* DCT-2$_n$, DST-2$_n$, DCT-4$_n$, DST-4$_n$ *have a mon-mon symmetry (χ, ψ) with nonzero matrix entries ± 1 arising from the group of automorphisms G_n of $\mathbb{C}[x]/T_n$ (cf. Theorem 9.5). Denote by $Z_n = \langle \sigma \mid \sigma^n = 1 \rangle$ the cyclic group of order n. The symmetry group for* DCT-2$_n$, DST-2$_n$ *is Z_n; for* DCT-4$_n$, DST-4$_n$ *it is Z_{2n}. The respective monomial representation χ is given by*

$$
\begin{aligned}
&\text{DCT-2}_n: && \sigma \mapsto (T_i \mapsto T_{ki} \bmod T_n)^{-1},\\
&\text{DST-2}_n: && \sigma \mapsto (U_i \mapsto U_{k-1+ki} \bmod T_n)^{-1},\\
&\text{DCT-4}_n: && \sigma \mapsto (V_i \mapsto V_{(k-1)/2+ki} \bmod T_n)^{-1},\\
&\text{DST-4}_n: && \sigma \mapsto (W_i \mapsto W_{(k-1)/2+ki} \bmod T_n)^{-1},
\end{aligned}
\tag{9.5}
$$

where $i = 0, \ldots, n-1$, and $k = 3$ for $n = 4$, and $k = 5$ for $n \geq 8$.

Proof. Let $g_k \in G_n$, i.e., $T_1^{g_k} = T_k$. We consider the first case DCT-2$_n$ = DCT-3$_n^T$ with associated algebra $A = \mathbb{C}[x]/T_n$ and $b = (T_0, \ldots, T_{n-1})$, i.e., DCT-2$_n$ decomposes the right regular representation of A (Theorem 6.2). Following Theorem 9.5, we have to find a monomial base change matrix $M_{g_k} : b \to b'$ such that T_1 operates on b as $T_1^{g_k} = T_k$ on b'. This is afforded by $b' = (T_{k\cdot 0}, T_{k\cdot 1}, \ldots, T_{k\cdot n-1})$, since, using Lemma 4.1(ii),

T_1 on b			T_k on b'		
$T_1 \cdot T_0$	=	T_1	$T_1 \cdot T_{k\cdot 0}$	=	$T_{k\cdot 1}$
$T_1 \cdot T_i$	=	$(T_{i-1} + T_{i+1})/2$	$T_1 \cdot T_{ki}$	=	$(T_{k(i-1)} + T_{k(i+1)})/2$
$T_1 \cdot T_{n-1}$	=	$T_{n-1}/2$	$T_1 \cdot T_{k(n-1)}$	=	$T_{k(n-1)}/2$

where $i = 2, \ldots, n-2$ and in the last line we used $T_n \equiv 0$, and thus $T_{kn} = T_k(T_n) \equiv 0$, since T_k is an odd function. The base change $b \to b'$ is given by the matrix $M_k : T_i \mapsto T_{ki}$, $i = 0, \ldots, n-1$, and thus

$$\phi^{g_k} = \phi^{M_k}.$$

(Note that we consider right representations, where ϕ is conjugated into $\phi^{M^{-1}}$ by a base change with matrix M.) As in the proof of Lemma 9.6, we see that M_k is monomial, since every T_{ki} can be reduced to a suitable $\pm T_\ell \bmod T_n$, $0 \leq \ell \leq n-1$. Theorem 9.5 establishes a mon-mon symmetry for (χ, ψ). It remains to show that the symmetry group is cyclic of order n. To this end we need the sequence of T_ℓ, $\ell \geq 0$, reduced mod T_n, given in the first row of Table 9.2. We see that $M_k^e = \mathrm{I}_n$ iff $T_{k^e i} \equiv T_i \bmod T_n$ ($i = 0, \ldots, n-1$) iff $k^e \equiv \pm 1 \bmod 4n$. As in the proof of Lemma 9.6, this shows that the maximum order $e = n$ is obtained for $k = 3$ if $n = 4$, and $k = 5$ if $n \geq 8$.

TABLE 9.2

For $P \in \{T, U, V, W\}$ the sequence of polynomials $P_0, \dots, P_{4n-1}$ (i.e., one period) reduced mod T_n. The vertical lines indicate multiples of n.

DCT-3 :	$T_0 \; \dots \; T_{n-1}$	$0 \; -T_{n-1} \; \dots \; -T_1$	$-T_0 \; \dots \; -T_{n-1}$	$0 \; T_{n-1} \; \dots \; T_1$
DST-3 :	$U_0 \; \dots \; U_{n-1}$	$U_{n-2} \; \dots \; U_0 \; 0$	$-U_0 \; \dots \; -U_{n-1}$	$-U_{n-2} \; \dots \; -U_0 \; 0$
DCT-4 :	$V_0 \; \dots \; V_{n-1}$	$-V_{n-1} \; \dots \; -V_0$	$-V_0 \; \dots \; -V_{n-1}$	$V_{n-1} \; \dots \; V_0$
DST-4 :	$W_0 \; \dots \; W_{n-1}$	$W_{n-1} \; \dots \; W_0$	$-W_0 \; \dots \; -W_{n-1}$	$-W_{n-1} \; \dots \; -W_0$

The proof of the other three cases is analogous. The base b is replaced by $b = (P_0, \dots, P_{n-1})$, where $P = U, V, W$, respectively.

The respective base change $b \to b'$ given by the matrix M_k corresponding to the automorphism g_k is given in lines 2–4 of (9.5) (without the inversion). The operation of T_k on b' can again be established using Lemma 4.1(ii). To determine the order of M_k we need, in each case, the sequence of P_ℓ reduced mod T_n given in Table 9.2. (Each of these sequences has period $4n$.) Surprisingly, it turns out that for the DCT-4 and DST-4, the symmetry group is a factor of 2 larger than G_n. We consider the example DCT-4_n. Denote by $V_{a_{e,i}}$ the image of V_i under M_k^e, $i = 0, \dots, n-1$. We get the recurrence and its solution

$$a_{0,i} = i, \quad a_{e,i} = (k-1)/2 + k \cdot a_{e-1,i} \quad \Rightarrow \quad a_{e,i} = (k^e - 1)/2 + ik^e.$$

Using the third row of Table 9.2, we get

$$\begin{aligned} & V_{(k^e-1)/2+ik^e} \equiv V_i \bmod T_n \quad (i = 0, \dots, n-1) \\ \Leftrightarrow \; & (k^e - 1)/2 + ik^e \equiv i \text{ or } -i-1 \bmod 4n \quad (i = 0, \dots, n-1) \\ \Leftrightarrow \; & k^e(2i+1) \equiv \pm(2i+1) \bmod 8n \quad (i = 0, \dots, n-1) \\ \Leftrightarrow \; & k^e \equiv \pm 1 \bmod 8n, \end{aligned}$$

which shows that the maximum order $e = 2n$ is obtained for $k = 3$ if $n = 4$, and $k = 5$ if $n \geq 8$. ∎

We conclude this section with a small example.

Example 9.8 (DCT-4, size 4). *Using Theorem* 9.7, *the* DCT-4_4 *has a mon-mon symmetry (χ, ψ) with a cyclic symmetry group $Z_8 = \langle \sigma \rangle$. The image $\chi(\sigma)$ is determined by the inverse of $V_i \mapsto V_{1+3i}$* mod T_4, *$i = 0, \dots, 3$. Using Table* 9.2, *we get $V_4 \equiv -V_3$, $V_7 \equiv -V_0$, $V_{10} \equiv -V_2$, and thus*

$$\chi(\sigma) = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 \end{bmatrix}, \quad \psi(\sigma) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

The matrix $\psi(\sigma)$ was computed using AREP. Both matrices have order 8.

The mon-mon symmetry of the DCT-4_4 given in Example 9.8 has been used in [21] to derive a fast algorithm (the symmetry is stated in a different way) and, using Theorem 8.6(iii), a fast algorithm for the DCT-2_8. The derivation essentially follows Figure 9.1, but the two monomial representations ϕ_1, ϕ_2 are decomposed over $\mathbb{Q}$. This concentrates all nonrational operations in the correction matrix D.

Remark. Using AREP, we have verified (up to a certain size) that all 16 types of DTTs possess mon-mon symmetries for every size n.

10. Other fast algorithms. The algebraic methods presented in sections 8 and 9 explain most of the algorithms from the literature. There is one class of algorithms, however, that cannot be explained by the methods presented so far. We will briefly discuss these algorithms to make this paper a comprehensive overview on DTT algorithms.

In short, it is possible to compute DTTs by embedding the transform matrix into a larger transform that can be computed efficiently. As an example, we consider the first algorithm proposed for the DCT-$2_n = [\cos k(\ell + \frac{1}{2})\pi/n]$ (see [1]). If we define the DFT by

$$\mathrm{DFT}_n = [e^{2\pi ikl/n}]_{k,\ell=0,\dots,n-1},$$

we can readily derive

$$\mathrm{re}(\mathrm{diag}_{k=0}^{2n-1}(e^{\pi ik/2n}) \cdot \mathrm{DFT}_{2n}) = \left[\cos k\left(\ell + \frac{1}{2}\right)\pi/n\right]_{k,\ell=0,\dots,2n-1},$$

where $\mathrm{re}(M)$ denotes the real part of the matrix M. This shows that a DCT-2_n can be computed by padding an input vector x of length n with n zeros, followed by multiplying with a scaled DFT of size $2n$. The first n entries contain the result.

Similar constructions allow the computation of each DTT via a DFT of appropriate length. This shows that the arithmetic complexity of each DTT_n is $O(n \log n)$, independent of the size n. In particular, this includes the DTTs of types 5–8, for which no other algorithms exist in the literature.

Embeddings into other transforms are also possible. For example, Theorem 8.7 allows us to embed a DTT of type 5–8 into a DTT of type 1 or 2.

11. Summary. We have given a complete characterization of all 16 types of DTTs as scaled polynomial transforms corresponding to appropriate A-modules M with basis b, where $A = \mathbb{C}[x]/p(x)$, $M = f \cdot A$ with a scaling function f, and b is a sequence of Chebyshev polynomials (Theorem 6.2). Every DTT is uniquely determined by this algebraic property.

We then used the algebraic characterization to derive by algebraic means most of the fast DTT algorithms known in the literature, and identified the mathematical principles behind each algorithm. In particular we derived the following:

1. *Algorithms by direct manipulation/decomposition of M* (section 8): (a) Translation between DTTs by duality (Theorem 8.2), (b) translation between DTTs by base change (Theorem 8.5), (c) decomposition by polynomial factorization (Theorems 8.6 and 8.7), (d) decomposition by polynomial decomposition (Theorem 8.8).
2. *Algorithms by group symmetries* (section 9): (a) Decomposition by mon-irred symmetry (Theorem 9.3), (b) decomposition by mon-mon symmetry (Theorem 9.5).
3. *Algorithms by embedding* (section 10).

Our results show clearly that the connection between digital signal processing and the representation theory of algebras goes beyond the DFT. The question that remains is to what extent this connection can be extended to include other transforms and their fast algorithms and how this connection can be exploited for applications in signal processing. We want to conclude by posing this question: *To what extent is signal processing algebraic?*

Appendix. Orthonormal DCTs and DSTs. Table A.1 gives the orthonormal versions of the 16 DTTs.

TABLE A.1

Definition of the orthonormal versions of the DCTs and DSTs; $a_{k,l}$ is the entry at row k and column l of the respective unscaled DTT as given in Table 5.1. All matrices have size $(n \times n)$, with row index $k = 0, \ldots, n-1$ and column index $\ell = 0, \ldots, n-1$. The row/column scaling factors are given by $c_i = 1/\sqrt{2}$ for $i = 0$ and $= 1$ else; $d_i = 1/\sqrt{2}$ for $i = n-1$ and $= 1$ else.

	DCTs	DSTs
type 1	$\sqrt{\frac{2}{n-1}} \cdot c_k c_\ell d_k d_\ell \cdot a_{k,l}$	$\sqrt{\frac{2}{n+1}} \cdot a_{k,l}$
type 2	$\sqrt{\frac{2}{n}} \cdot c_k \cdot a_{k,l}$	$\sqrt{\frac{2}{n}} \cdot c_k \cdot a_{k,l}$
type 3	$\sqrt{\frac{2}{n}} \cdot c_\ell \cdot a_{k,l}$	$\sqrt{\frac{2}{n}} \cdot c_\ell \cdot a_{k,l}$
type 4	$\sqrt{\frac{2}{n}} \cdot a_{k,l}$	$\sqrt{\frac{2}{n}} \cdot a_{k,l}$
type 5	$\sqrt{\frac{2}{n-1/2}} \cdot c_k c_\ell \cdot a_{k,l}$	$\sqrt{\frac{2}{n+1/2}} \cdot a_{k,l}$
type 6	$\sqrt{\frac{2}{n-1/2}} \cdot c_k d_\ell \cdot a_{k,l}$	$\sqrt{\frac{2}{n+1/2}} \cdot a_{k,l}$
type 7	$\sqrt{\frac{2}{n-1/2}} \cdot d_k c_\ell \cdot a_{k,l}$	$\sqrt{\frac{2}{n+1/2}} \cdot a_{k,l}$
type 8	$\sqrt{\frac{2}{n+1/2}} \cdot a_{k,l}$	$\sqrt{\frac{2}{n-1/2}} \cdot d_k d_\ell \cdot a_{k,l}$

REFERENCES

[1] N. AHMED, T. NATARAJAN, AND K. R. RAO, *Discrete cosine transform*, IEEE Trans. Computers, C-23 (1974), pp. 90–93.

[2] L. AUSLANDER, E. FEIG, AND S. WINOGRAD, *Abelian semi-simple algebras and algorithms for the discrete Fourier transform*, Adv. in Appl. Math., 5 (1984), pp. 31–55.

[3] T. BETH, *Verfahren der Schnellen Fouriertransformation*, Teubner, Stuttgart, 1984.

[4] T. BETH, *On the computational complexity of the general discrete Fourier transform*, Theoret. Comput. Sci., 51 (1987), pp. 331–339.

[5] P. BÜRGISSER, M. CLAUSEN, AND M. A. SHOKROLLAHI, *Algebraic Complexity Theory*, Springer, Berlin, 1997.

[6] S. CHAN AND K. HO, *Direct methods for computing discrete sinusoidal transforms*, Radar Signal Process. IEE Proc. F, 137 (1990), pp. 433–442.

[7] W.-H. CHEN, C. SMITH, AND S. FRALICK, *A fast computational algorithm for the discrete cosine transform*, IEEE Trans. Comm., COM-25 (1977), pp. 1004–1009.

[8] T. S. CHIHARA, *An Introduction to Orthogonal Polynomials*, Gordon and Breach, New York, 1978.

[9] M. CLAUSEN, *Beiträge zum Entwurf schneller Spektraltransformationen*, Habilitationsschrift, Universität Karlsruhe, Karlsruhe, Germany, 1988.

[10] M. CLAUSEN AND U. BAUM, *Fast Fourier Transforms*, BI Wissenschafts-Verlag, 1993.

[11] J. W. COOLEY AND J. W. TUKEY, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297–301.

[12] W. C. CURTIS AND I. REINER, *Representation Theory of Finite Groups*, Interscience, New York, 1962.

[13] P. DIACONIS AND D. ROCKMORE, *Efficient computation of the Fourier transform on finite groups*, J. Amer. Math. Soc., 3 (1990), pp. 297–332.

[14] J. R. DRISCOLL, D. M. HEALY, JR., AND D. N. ROCKMORE, *Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs*, SIAM J. Comput., 26 (1997), pp. 1066–1099.

[15] S. EGNER, *Zur Algorithmischen Zerlegungstheorie Linearer Transformationen mit Symmetrie*, Ph.D. thesis, Universität Karlsruhe, Informatik, Karlsruhe, Germany, 1997.

[16] S. EGNER, J. JOHNSON, D. PADUA, M. PÜSCHEL, AND J. XIONG, *Automatic derivation and implementation of signal processing algorithms*, ACM SIGSAM Bulletin Communications in Computer Algebra, 35 (2001), pp. 1–19.

[17] S. EGNER AND M. PÜSCHEL, *AREP—Constructive Representation Theory and Fast Signal Transforms*, GAP software share package, 1998; available online at http://www.ece.cmu.edu/∼smart/arep/arep.html.

[18] S. EGNER AND M. PÜSCHEL, *Automatic generation of fast discrete signal transforms*, IEEE Trans. Signal Process., 49 (2001), pp. 1992–2002.

[19] S. Egner and M. Püschel, *Symmetry-based matrix factorization*, J. Symbolic Comput., to appear.
[20] E. Feig, *A fast scaled-DCT algorithm*, Proc. SPIE, 1244 (1990), pp. 2–13.
[21] E. Feig and S. Winograd, *Fast algorithms for the discrete cosine transform*, IEEE Trans. Signal Process., 40 (1992), pp. 2174–2193.
[22] The GAP Team, *GAP—Groups, Algorithms, and Programming*, University of St. Andrews, Scotland, 1997; available online at http://www-gap.dcs.st-and.ac.uk/∼gap/.
[23] M. Heideman, D. Johnson, and C. Burrus, *Gauss and the history of the fast Fourier transform*, Archive for History of Exact Sciences, 34 (1985), pp. 265–277.
[24] H. Hou, *A fast recursive algorithm for computing the discrete cosine transform*, IEEE Trans. Acoustics, Speech, and Signal Processing, ASSP-35 (1987), pp. 1455–1461.
[25] N. Jacobson, *Basic Algebra* II, W. H. Freeman, San Francisco, 1980.
[26] T. Kailath and V. Olshevsky, *Displacement structure approach to discrete trigonometric transform based preconditioners of G. Strang and T. Chan type*, Calcolo, 33 (1996), pp. 191–208.
[27] H. Kitajima, *A symmetric cosine transform*, IEEE Trans. Computers, C-29 (1980), pp. 317–323.
[28] B. Lee, *A new algorithm to compute the discrete cosine transform*, IEEE Trans. Acoustics, Speech, and Signal Processing, ASSP-32 (1984), pp. 1243–1245.
[29] D. Maslen and D. Rockmore, *Generalized FFTs—A survey of some recent results*, in Proceedings of the DIMACS Workshop on Groups and Computation, Rutgers University, 1995, vol. 28, pp. 182–238.
[30] J. C. Mason, *Chebyshev polynomials of the second, third and fourth kind in approximation, indefinite integration, and integral transforms*, J. Comput. Appl. Math., 49 (1993), pp. 169–178.
[31] T. Minkwitz, *Algorithmensynthese für lineare Systeme mit Symmetrie*, Ph.D. thesis, Universität Karlsruhe, Informatik, Karlsruhe, Germany, 1993.
[32] T. Minkwitz, *Algorithms Explained by Symmetry*, Lecture Notes in Comput. Sci. 900, 1995, pp. 157–167.
[33] J. M. F. Moura and M. G. S. Bruno, *DCT/DST and Gauss-Markov fields: Conditions for equivalence*, IEEE Trans. Signal Process., 46 (1998), pp. 2571–2574.
[34] D. Potts and G. Steidl, *Optimal trigonometric preconditioners for nonsymmetric Toeplitz system*, Linear Algebra Appl., 281 (1998), pp. 265–292.
[35] D. Potts, G. Steidl, and M. Tasche, *Fast algorithms for discrete polynomial transforms*, Math. Comp., 67 (1998), pp. 1577–1590.
[36] M. Püschel, *Konstruktive Darstellungstheorie und Algorithmengenerierung*, Ph.D. thesis, Universität Karlsruhe, Informatik, 1998; also available in English as Technical report Drexel-MCS-1999-1, Drexel University, Philadelphia, 1999.
[37] M. Püschel, *Decomposing monomial representations of solvable groups*, J. Symbolic Comput., 34 (2002), pp. 561–596.
[38] M. Püschel and J. M. F. Moura, *The discrete trigonometric transforms and their fast algorithms: An algebraic symmetry approach*, in Proceedings of the 10th IEEE DSP Workshop, Pine Mountain, GA, 2002.
[39] C. M. Rader, *Discrete Fourier transforms when the number of data samples is prime*, Proc. IEEE, 56 (1968), pp. 1107–1108.
[40] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*, Academic Press, Boston, 1990.
[41] M. O. Rayes, V. Trevisan, and P. S. Wang, *Factorization of Chbeyshev Polynomials*, Technical report ICM-199802-0001, Kent State University, Kent, OH, 1998.
[42] T. J. Rivlin, *The Chebyshev Polynomials*, Wiley Interscience, New York, 1974.
[43] D. Rockmore, *Efficient computation of Fourier inversion for finite groups*, Assoc. Comp. Mach., 41 (1994), pp. 31–66.
[44] D. Rockmore, *Some applications of generalized FFT's*, in Proceedings of DIMACS Workshop in Groups and Computation, vol. 28, 1995, pp. 329–370.
[45] V. Sánchez, P. García, A. M. Peinado, J. C. Segura, and A. J. Rubio, *Diagonalizing properties of the discrete cosine transforms*, IEEE Trans. on Signal Process., 43 (1995), pp. 2631–2641.
[46] G. Steidl and M. Tasche, *A polynomial approach to fast algorithms for discrete Fourier-cosine and Fourier-sine transforms*, Math. Comp., 56 (1991), pp. 281–296.
[47] G. Strang, *The discrete cosine transform*, SIAM Rev., 41 (1999), pp. 135–147.
[48] G. Szegö, *Orthogonal Polynomials*, 3rd ed., Amer. Math. Soc. Colloq. Publ. 23, AMS, Providence, RI, 1967.

[49] R. Tolimieri, M. An, and C. Lu, *Algorithms for Discrete Fourier Transforms and Convolution*, 2nd ed., Springer, New York, 1997.
[50] M. Vetterli and H. Nussbaumer, *Simple FFT and DCT algorithms with reduced number of operations*, Signal Process., 6 (1984), pp. 267–278.
[51] Z. Wang, *Reconsideration of "A fast computational algorithm for the discrete cosine transform,"* IEEE Trans. Comm., COM-31 (1983), pp. 121–123.
[52] Z. Wang, *Fast algorithms for the discrete W transform and for the discrete Fourier transform*, IEEE Trans. Acoustics, Speech, and Signal Processing, ASSP-32 (1984), pp. 803–816.
[53] Z. Wang and B. Hunt, *The discrete W transform*, Appl. Math. Comput., 16 (1985), pp. 19–48.
[54] S. Winograd, *Arithmetic Complexity of Computations*, CBMS-NSF Regional Conf. Ser. in Appl. Math. 33, SIAM, Philadelphia, 1980.
[55] P. Yip and K. Rao, *A fast computational algorithm for the discrete sine transform*, IEEE Trans. Comm., COM-28 (1980), pp. 304–307.
[56] P. Yip and K. Rao, *Fast decimation-in-time algorithms for a family of discrete sine and cosine transforms*, Circuits Systems Signal Process., 3 (1984), pp. 387–408.
[57] P. Yip and K. Rao, *The decimation-in-frequency algorithms for a family of discrete sine and cosine transforms*, Circuits Systems Signal Process., 7 (1988), pp. 3–19.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1317–1337

APPROXIMATION SCHEMES FOR MINIMUM LATENCY PROBLEMS*

SANJEEV ARORA† AND GEORGE KARAKOSTAS‡

Abstract. The *minimum latency problem*, also known as the traveling repairman problem, is a variant of the traveling salesman problem in which the starting node of the tour is given and the goal is to minimize the sum of the *arrival times* at the other nodes. We present a quasi-polynomial time approximation scheme (QPTAS) for this problem when the instance is a weighted tree, when the nodes lie in $\mathbb{R}^d$ for some fixed d, and for planar graphs. We also present a polynomial time constant factor approximation algorithm for the general metric case. The currently best polynomial time approximation algorithm for general metrics, due to Goemans and Kleinberg, computes a 3.59-approximation.

Key words. minimum latency tour, traveling repairman, search ratio, randomized search ratio, vehicle routing, quasi-polynomial approximation schemes, approximation algorithms

AMS subject classifications. 68W25, 90B06, 05C38

DOI. 10.1137/S0097539701399654

1. Introduction.

The *minimum latency problem*, also known as the *deliveryman* or *traveling repairman* problem (Afrati et al. [1], Minieka [16], Lucena [15], Bianco, Mingossi, and Ricciardelli [9]), is a variant of the traveling salesman problem (TSP) in which the starting node of the tour is given and the goal is to minimize the sum of the arrival times or *latencies* at the other nodes. (The *latency* of a node is the distance covered before reaching that node.) This natural combinatorial problem arises in many day-to-day situations, whenever a server (e.g., a repairman or a disk head) has to accommodate a set of requests (each represented by a point) so as to minimize their total (or average) waiting time. The tour that achieves this goal will henceforth be called the *minimum latency tour* (MLT).

More formally, the problem is defined as follows.

MINIMUM LATENCY TOUR

Input: A set of n points (one of them designated as the starting point p_1), and a symmetric distance matrix $[d_{ij}]$.

Output: A tour $p_1 \to p_2 \to \cdots \to p_n$ that visits *all* points and minimizes the total latency

$$\sum_{i=2}^{n} \sum_{j=1}^{i-1} d_{p_j, p_{j+1}}. \tag{1.1}$$

*Received by the editors December 13, 2001; accepted for publication (in revised form) May 21, 2003; published electronically August 15, 2003. An extended abstract appeared in *Proceedings of the 31st ACM Symposium on Theory of Computing*, Atlanta, GA, 1999, pp. 688–693. This research was supported by an NSF CAREER award, NSF grants CCR-9502747 and CCR-0098180, an Alfred Sloan Fellowship, and a Packard Fellowship.

http://www.siam.org/journals/sicomp/32-5/39965.html

†Department of Computer Science, Princeton University, Princeton, NJ 08544 (arora@cs.princeton.edu).

‡Department of Computing and Software, McMaster University, Hamilton, Ontario L8S 4K1, Canada (karakos@mcmaster.ca).

By rearrangement, the objective function (1.1) can be rewritten as a weighted sum:

$$\sum_{i=1}^{n-1}(n-i)d_{p_i,p_{i+1}}. \tag{1.2}$$

Here we study the restriction of the problem to distance matrices that define a metric; i.e., the distances satisfy the triangle inequalities. From the objective function formulation (1.2) it becomes apparent that the MLT problem is a weighted variation of the TSP (where the objective function to be minimized by the tour is $\sum_{i=1}^{n-1} d_{p_i,p_{i+1}}$). However, it has a reputation for being much harder than the TSP.

For example, the MLT does not possess the locality property of the TSP for local changes in the structure of a metric space. Even a very small local change of a tour can affect the arrival time of *all* the points visited by the tour afterwards, and it may change the value of (1.2) by too much. On the other hand, local changes affect the *length* of the tour only locally. This lack of locality is conceivably prohibitive for the design of "divide and conquer" algorithms. This is not the case, for example, for the Euclidean TSP, where recent "divide and conquer" algorithms by Arora [4] (and, independently, Mitchell [17]) solve the problem almost optimally.

It is easy to prove (Blum et al. [10]) that calculating the MLT is at least as hard as calculating the TSP: Given an instance of the TSP, create an instance for the MLT by adding m new points (for a large m) at infinity (i.e., far enough away from the points of the TSP instance). Then the MLT on the augmented set of points will have to follow the TSP tour on the original points before visiting the new ones. This reduction proves the NP-hardness of the MLT for all the metric spaces where TSP is NP-hard. But even for metrics where the TSP is solvable trivially, the computation of the MLT may be hard. Such is the case for tree metrics (the points are vertices of a tree with nonnegative edge lengths, and the distance between two points is the length of the unique tree path that connects them). MLT was recently proven to be NP-hard even for tree metrics [19].

Afrati et al. [1] give a simple dynamic programming algorithm that solves the problem when the points are on a line in polynomial time. Minieka [16] and Blum et al. [10] prove that in the case of *unweighted* trees (trees whose edges have unit length), any depth-first walk on the tree is an optimal MLT. Minieka [16] gives also an exponential time algorithm that solves the problem exactly for any weighted tree. For general metric spaces, Bianco, Mingossi, and Ricciardelli [9] and Lucena [15] give exponential time algorithms that find a MLT.

The general metric case of the latency problem is MAX-SNP-hard (this follows from the reduction that proves the MAX-SNP-hardness of TSP with distances either 1 or 2 by Papadimitriou and Yannakakis [18]), and therefore the results of Arora et al. [8] imply that unless P = NP, a polynomial time approximation scheme (PTAS) does not exist. Blum et al. [10] gave a 144-approximation algorithm for the metric case and an 8-approximation for weighted trees. Goemans and Kleinberg [12] then gave a 21.55-approximation in the metric case. The Goemans–Kleinberg algorithm requires as a subroutine a good approximation algorithm for the k-TSP problem ("given n nodes and a number k, find the shortest salesman tour containing k nodes"). Their algorithm achieves a 3.59-approximation ratio for the tree case and a $3.59 \times$ (*appr. ratio for k-MST*) approximation ratio for general metric spaces. Currently, the best approximation ratio for the general metric k-MST is $2+\varepsilon$ for any constant $\varepsilon > 0$ due to Arora and Karakostas [7] and $1+\varepsilon$ for any $\varepsilon > 0$ for constant-dimensional Euclidean spaces due to Arora [4] (for planar instances, see also Mitchell [17]). These

can be used to improve the approximation ratio achieved by the Goemans–Kleinberg algorithm in these cases, achieving a 7.18-approximation in the metric case and a 3.59-approximation in the Euclidean case. Subsequently to the first appearance of our work, Archer and Williamson [3] gave a faster approximation algorithm for the MLT (although their approximation factor is somewhat worse than the Goemans–Kleinberg one), and Fakcharoenphol, Harrelson, and Rao [11] present approximation algorithms for the generalization of the traveling repairman with k repairmen.

We present a new and very simple technique that leads to *approximation schemes* for minimum latency for all weighted trees and constant-dimensional Euclidean spaces. To compute a $(1+\varepsilon)$-approximation for the problem on n nodes, the algorithm requires $n^{O(\log^2 n/\varepsilon)}$ time on weighted trees and $n^{O(\log n/\varepsilon)}$ time in $\mathbb{R}^2$. We also present an 11.656-approximation in the metric case. Though this approximation ratio is worse than that of the algorithm by Goemans and Kleinberg [12], our algorithm seems to be simpler.

Previous papers have taken the approach of computing solutions to a sequence of k-TSP instances, for $k = 1, 2, \ldots, n$, and concatenating some of these tours to get the final tour. The main intuition in these algorithms may be described as follows: Visit the nodes closest to the start node as soon as possible. The main idea in our algorithm is easier to state. The algorithm finds the tour as a union of $O(\log n/\varepsilon)$ tours, containing $n_1, n_2, \ldots$ nodes. The important difference is that the choice of $n_1, n_2, \ldots$, does not depend on the instance; it depends only on n, ε. (Thus, as noted in section 3, our algorithm can be viewed as an approximation-preserving reduction to a version of vehicle routing.) In fact, in our algorithm for the general metric case, the first salesman tour contains more than $n/2$ nodes. This seems to go against the received intuition of "visit the nodes closest to the start node first." Conceivably, our technique could be combined with that intuition to get better algorithms, but we do not currently know how.

1.1. MLT and the randomized search ratio. Koutsoupias, Papadimitriou, and Yannakakis [14] consider a graph exploration problem in which an explorer is presented with a weighted graph on n nodes. One of the nodes contains a treasure, which the explorer will recognize only when he sees it. The goal is to design a walk on the graph such that the explorer arrives at the treasure as quickly as possible. The *search ratio* of the graph is the worst-case ratio of the arrival time and the distance of the treasure to the start node. The *randomized search ratio* is defined similarly, except the walk may be randomized, and so we need the *expected* arrival time at the treasure.

More formally, the two problems are defined as follows.

SEARCH RATIO

Input: A graph G with distances on edges, and a root vertex r.

Output:

$$\sigma(G, r) = \min_{\pi} \max_{v \in G} \frac{d_\pi(r, v)}{d(r, v)},$$

where $d(r, v)$ denotes the distance from r to v and $d_\pi(r, v)$ denotes the distance from r to v in the walk π.

RANDOMIZED SEARCH RATIO

Input: A graph G with distances on edges, and a root vertex r.

Output:

$$\rho(G,r) = \min_{\Delta} \max_{v \in G} \frac{E_{\Delta}[d_{\pi}(r,v)]}{d(r,v)},$$

where Δ ranges over *distributions of walks* and $E_{\Delta}[d_{\pi}(r,v)]$ denotes the *expected* distance from r to v in the walk π, when π is drawn randomly according to distribution Δ.

As shown in Koutsoupias, Papadimitriou, and Yannakakis [14], computing the search ratio and the randomized search ratio of a graph G with respect to a root node r is NP-complete and MAX-SNP-hard. They also show that the minimum latency problem is the polyhedral separation problem of the dual of the randomized search ratio problem. Indeed, the randomized search ratio can be expressed as the solution of the following $(n+1) \times n!$ linear program:

$$\begin{aligned}
&\min\ \rho \ \text{ s.t.}\\
&\sum_{\text{walk } \pi} x_{\pi} d_{\pi}(r,v) \le \rho \cdot d(r,v) \quad \forall v \in V,\\
&\sum_{\text{walk } \pi} x_{\pi} = 1,\\
&x_{\pi} \ge 0.
\end{aligned}$$

Its dual is

$$\begin{aligned}
&\min \sum_{v} d(r,v) y_v - z \ \text{ s.t.}\\
&\sum_{v} d_{\pi}(r,v) y_v - z \ge 0 \quad \forall \text{ walk } \pi,\\
&y_v \ge 0.
\end{aligned}$$

To solve the dual by using the ellipsoid algorithm using the general framework of Grötschel, Lovász, and Schrijver [13], we should be able to check whether a given point $(\vec{y}, z) \in \mathbb{R}^{n+1}$ lies in a feasible solution or not. In case it is *not* feasible, we need a violated inequality; i.e., we should be able to decide whether $\min_{\pi} \sum_{v} d_{\pi}(r,v) y_v \le z$. Koutsoupias, Papadimitriou, and Yannakakis [14] observe that this decision problem is *polynomially equivalent* to the MLT problem, under some very mild restrictions that do not affect approximability.

Using the general framework for convex optimization in Grötschel, Lovász, and Schrijver [13], É. Tardos has observed that if the minimum latency problem has a PTAS for a certain class of metrics, then the randomized search ratio problem has an approximation scheme for that same class of metrics (Koutsoupias, Papadimitriou, and Yannakakis [14]). Thus our algorithm implies the existence of a quasi-polynomial time approximation scheme (QPTAS) for the randomized search ratio for trees and Euclidean spaces. We do not know of a previous use of the framework in Grötschel, Lovász, and Schrijver [13] to design an approximation scheme.

2. The main idea: Local structure does not matter. It is well known that minimizing the total tour length may give a tour of very high latency. In this section we show that the strategy of minimizing tour lengths works, as long as it is done in a local fashion. Namely, to find a $(1+\varepsilon)$-approximate MLT, it suffices to find the tour as a union of $O(\log n/\varepsilon)$ segments, where the number of nodes in successive segments

decreases geometrically. Within each segment the order of visits to the nodes does not matter, as long as the total length is close to minimum. Of course, we have not specified thus far how to partition nodes into the segments in the correct way. In the Euclidean and tree-metric cases, we can do this with a simple dynamic programming. In the metric case, we recourse to a greedy strategy that introduces another source of suboptimality.

We note that the idea of finding a low latency tour as a union of salesman tours/paths is present in all earlier papers. However, those earlier strategies decided in an adaptive fashion which salesman tours to combine. In contrast, our algorithm can decide at the very start how many nodes must be present in each salesman path. We prove that we can break an MLT tour into segments so that local changes within a segment does not affect the total latency by much, and then replace each segment by an optimum salesman path, so that the new tour is still near optimal.

Let $\mathcal{T}$ be an optimal tour with total latency OPT. Let $\varepsilon > 0$ be any parameter. Break this tour into $k = O(\log n/\varepsilon)$ segments so that in segment i we visit n_i nodes, where

$$n_i = (1+\varepsilon)^{k-1-i} \text{ for } i = 1, \ldots, k-1,$$
$$n_k = 1/\varepsilon.$$

Depending on n and ε, some n_i may not be integers. This means that some nodes will be "broken" in two fractions, belonging to two segments. Since for our calculations up to section 4.6 the start node of a segment is the finish node of the previous segment, it does not matter what fraction of this node belongs to each segment. Let the length of the ith segment be T_i. If we let $n_{>i}$ denote the total number of nodes visited in segments numbered $i+1$ and later, then a simple calculation shows that (and this was the reason for our choice of n_i's)

$$n_{>i} = \sum_{j>i} n_j = \frac{n_i}{\varepsilon} \text{ for every } i = 1, \ldots, k-1. \tag{2.1}$$

Now imagine doing the following in each segment except the last one: replace that segment by the minimum-cost traveling salesman path on the same subset of nodes while maintaining the starting and ending points. We claim that the new latency is at most $(1+\varepsilon)OPT$. First, note that $\sum_{j=1}^{m-1} T_j$ is a lower bound on the latency of any node in the mth segment. Adding over all segments, we get the following lower bound on OPT:

$$OPT \geq \sum_{i=1}^{k-1} n_{>i} \cdot T_i. \tag{2.2}$$

Consider the effect of replacing the ith segment with the shortest salesman path on that subset of nodes. The length of the segment cannot increase, and thus neither can the latency of nodes in later segments. The latency of nodes within the segment can only rise by $n_i T_i$. Thus the new latency is at most the lower bound in (2.2) plus

$$\sum_{i=1}^{k-1} n_i \cdot T_i. \tag{2.3}$$

Now condition (2.1) implies that the new latency is at most $(1+\varepsilon)OPT$, as claimed, and the proof of the main idea is complete.

THEOREM 2.1. *Let OPT be the total latency of the MLT. There exists a tour that is a concatenation of* $O(\frac{\log n}{\varepsilon})$ *optimal salesman paths and whose total latency is at most* $(1+\varepsilon)OPT$.

The importance of this structure theorem for the design of approximation schemes is apparent for cases where the TSP is efficiently solvable (e.g., weighted trees). All one has to do is to decide which node goes into which segment and then calculate the optimal salesman path for each segment. Even if, instead of the optimum salesman path in each segment, we use a $(1+\gamma)$-approximate salesman path, then the latency of the tour of Theorem 2.1 is $(1+\gamma\cdot\varepsilon+\gamma+\varepsilon)OPT$. This is the case, for example, for Euclidean spaces of fixed dimension, where there are approximation schemes for the TSP (cf. Arora [4]). For these cases the difficulty lies in the placement of nodes into the appropriate segments. This has the flavor of a vehicle routing problem, and this will become more precise in the following section.

3. Reduction from minimum latency to weighted vehicle routing. The purpose of this section is to note that our technique described above implies a quasi-polynomial time approximation-preserving reduction from minimum latency to a version of weighted vehicle routing with *per-mile costs*. We do not know of a prior result along these lines.

WEIGHTED VEHICLE ROUTING WITH PER-MILE COSTS

Input: A set of n clients, who have to be visited by a fleet of m vehicles. Vehicle i has a designated depot s_i at which to start and another depot t_i at which to finish. It also has a *capacity* (which is the number of clients it can visit) c_i and a *per-mile cost* d_i.

Output: Assign clients to vehicles so as to respect the capacity constraints and minimize the total cost, which is the sum of distances covered by the vehicles, weighted by the per-mile cost

$$\sum_{i=1}^{m} d_i T_i,$$

where T_i is the distance traveled by vehicle i.

Suppose we are given an oracle for this version of vehicle routing. We can use this oracle to solve the MLT problem. Given a set of n nodes, the reduction proceeds as follows, where $k, n_i, n_{>i}$ have the same meaning as in section 2: "Let p_0 denote the starting node of the MLT. For every sequence of k nodes $p_1, p_2, \ldots, p_k$, use the vehicle routing oracle to construct a solution to the instance in which there are k vehicles, and the capacity of the ith vehicle is n_i, its per-mile cost is $n_{>i}$, and its start and end depots are p_i and p_{i+1} respectively. At the end, output the lowest cost solution found (over the choice of all sequences of k points)."

Clearly, if the vehicle routing oracle computes a ρ-approximation in polynomial time, our reduction will lead to a $\rho(1+\varepsilon)$-approximate solution for MLT in $n^{O(\log n/\varepsilon)}$ time. This follows from Theorem 2.1, since it suffices to go over all possible sequences $p_1, p_2, \ldots, p_k$, and this increases the running time by a factor of at most $O(n^k) = n^{O(\frac{\log n}{\varepsilon})}$.

4. Approximation algorithms for the MLT. The essence of Theorem 2.1 is that there is a near optimal latency tour of a potentially simpler structure, since it is the collection of just a few optimal salesman paths. We take advantage of this simpler structure in order to compute a near optimal tour for the cases of trees, Euclidean (and other norm) spaces, and planar graphs. The algorithms for these

cases are quasi-polynomial time *approximation schemes*. We also give a constant factor polynomial time approximation algorithm for the general metric case as an immediate result of Theorem 2.1. Although its approximation factor is somewhat worse than the approximation factor achieved by Goemans and Kleinberg [12], it seems simpler.

4.1. The tree case. The optimum salesman tour on a tree may in general need to visit a node an unbounded number of times. For example, in a star graph it must visit the center node $n-1$ times. However, the tour never needs to visit an edge more than twice, as is easily checked. Thus an optimum tour has very simple structure and can be found by depth-first search.

A MLT, on the other hand, could have a very complicated structure. Consider, for example, a complete binary tree in which all edges have zero weight except those attached to leaves. The MLT will first visit all the internal nodes in some arbitrary order, and then all the leaf nodes in sorted order by weight, thus crossing the root node potentially $n/2$ times.

However, our technique from section 2 allows us to view a near-optimum latency tour as a union of $O(\log n/\varepsilon)$ salesman paths. By observing that within each salesman path an edge is only visited twice, we can then ensure that each edge is visited only $O(\log n/\varepsilon)$ times overall. This idea underlies the proof of our *structure theorem* below and the approximation scheme for trees.

4.1.1. The structure theorem for trees. In this subsection we prove that there is a tour on a weighted tree whose latency is near optimal but crosses each node of the tree only a few times.

DEFINITION 1. *An $\alpha : \beta$-partition of a tree $\mathcal{T}$ with n nodes is the recursive partition of $\mathcal{T}$ into two subtrees with a common root so that for each subtree*

$$\alpha\ n \leq \textit{(size of subtree)} \leq \beta\ n.$$

The partition is recursive, and each separator can be thought as belonging to both subtrees it separates. At the bottom level of the recursion there are pairs of nodes connected by an edge.

It is easy to show that one can always find a $\frac{1}{3} : \frac{2}{3}$-partition of a tree.

LEMMA 4.1. *In any tree we can always find a node with the following property: Let $m \geq 2$ denote its degree and $f_1, \ldots, f_m$ denote the sizes of the subtrees attached to the node. There exists a subset $S \subseteq \{1, \ldots, m\}$ such that*

$$\left\lfloor \frac{n}{3} \right\rfloor \leq \sum_{i \in S} f_i \leq \left\lceil \frac{2n}{3} \right\rceil.$$

Proof. Start by picking a node of the tree. If there is a subtree with more than $\lceil \frac{2n}{3} \rceil$ nodes pick the root of this subtree and continue. If there is a subtree with number of nodes between $\lfloor \frac{n}{3} \rfloor$ and $\lceil \frac{2n}{3} \rceil$, then the current node satisfies the lemma requirements. If all subtrees have less than $\lfloor \frac{n}{3} \rfloor$ nodes (obviously, $m \geq 3$ in this case) it is easy to see that the lemma holds for the current node. ∎

We designate the node guaranteed to exist by Lemma 4.1 as a *separator node* and the $|S|$ components as the *left* of the tree and the other $m - |S|$ as the *right* of the tree. (The node itself is copied twice and appears in both sides.) Then we recur on the two sides. This gives a recursive partition of the tree. We say that a tour *crosses* the separator node if it goes from the left side to the right. Notice that a node may be visited many times before it is crossed.

It is obvious that since a minimum salesman tour is obtained by depth-first search, it needs to cross each separator node only twice. The combination of this fact with our main idea gives the following.

THEOREM 4.2 (structure theorem for weighted trees). *The following is true for every integer $n > 0$ and every $\varepsilon > 0$: For every weighted tree on n nodes with a node–separator-based partition as defined above, a tour exists with latency at most $(1+\varepsilon)OPT$ that crosses each separator node only $O(\log n/\varepsilon)$ times.*

Proof. Let $\mathcal{T}$ be the optimum tour. Divide it into $O(\log n/\varepsilon)$ segments as in section 2 and replace each segment by the optimum salesman path. Now use the fact that a minimum salesman path does not cross a separator node going from the left side to the right (or vice versa) of the partition at that node more than twice. ∎

4.1.2. The algorithm for trees. A simple dynamic programming approach that relies on the structure theorem, Theorem 4.2 can be used to compute a $(1+\varepsilon)$-approximate latency tour for general weighted trees.

ALGORITHM 1.

1. Identify a recursive $\frac{1}{3} : \frac{2}{3}$-partition of the tree.
2. Identify the node-separator at the top level of the partition.
3. "Guess" the number of times the tour crosses this node, and for each crossing, the length of the tour portion after the crossing and the number of nodes on that portion.
4. Keep track of the "guess" and recur on each side of the node-separator; i.e., compute a collection of paths that visits *all* the nodes on each side of the separator, is consistent with the "guess" in terms of lengths and number of nodes visited, and achieves minimum latency among all such paths.
5. Store the cost (latency) of the combined (at the separator) subtour in the "guess" entry of the dynamic programming table and return. ∎

By "guessing" in step 3 we refer to the exhaustive enumeration of all possible values for length and number of nodes for *each crossing* of the separator. Since the subtours on the two sides of the separator are the optimum subtours that are consistent with our "guess," their combination will be the optimum one that is consistent with this "guess." In the end of the enumeration, the algorithm will have created a collection of candidate solutions (one for each set of values for our "guesses"). Its output will be the tour with the minimum total latency. One of these tours calculated by the algorithm must be the near optimal tour guaranteed by the structure theorem, Theorem 4.2, since the algorithm is bound to encounter the specific set of "guesses" defined by it due to exhaustive enumeration. Hence the tour it produces is at least as good as this tour, and the algorithm is indeed an *approximation scheme*. The only thing remaining to prove is that it is a *quasi-polynomial* approximation scheme.

The running time of the algorithm is obviously dominated by the number of possible "guesses." The number of crossings through a node is at most $O(\log n/\varepsilon)$ (Theorem 4.2), and the number of nodes visited between two crossings cannot be greater than n. But notice that the length of the tour between two crossings can be exponential in the size of the input. This is the case when an edge has weight exponential in the size of the input. Then the number of guesses is also exponential, and the algorithm runs in exponential time, instead of quasi-polynomial. In order to get around this problem, we *round* the given instance by running the following **rounding procedure**:

1. Let L be the length of the longest path in the tree and $\delta > 0$ any constant smaller than ε. Merge (by contracting edges) all pairs of nodes with internode

distance at most $\delta L/n^2$.
2. Round each edge weight to its closest multiple of $\delta L/n^2$.
3. Divide each edge weights by $\delta L/n^2$.

After solving the problem on the rounded instance, we reinstate the merged edges to output the tour computed on the original instance.

LEMMA 4.3. *Let OPT be the optimal latency in the given instance, and L, δ are as in the above procedure. Then the MLT on the rounded instance and OPT differ by at most δ OPT. Moreover, the maximum internode distance in the rounded instance is at most $O(n^2/\delta) = O(n^2/\varepsilon)$.*

Proof. The second part of the lemma is obvious. Also, it is easy to see that the latency of each node in the original MLT has not changed by more than $O(\delta L/n)$, for a total change of $O(\delta L) = \delta OPT$. Notice that when we reinstate the merged edges, the latency increase cannot be more that δOPT. This means that we need to compute an $(1+\varepsilon-\delta)$-approximation tour on the rounded instance instead of a $(1+\varepsilon)$-approximation. But we certainly can do that since δ is an arbitrary constant. □

Thus if we run Algorithm 1 on the new rounded instance, the running time is quasi-polynomial.

THEOREM 4.4. *Algorithm 1 runs in time $n^{O(\log^2 n/\varepsilon)}$ and computes a tour whose latency is at most $(1+\varepsilon)$ OPT.*

Proof. If δ is the constant in the rounding procedure, then we apply Theorem 4.2 so that the approximation factor for the rounded instance is $1+\varepsilon-\delta$. Because of Lemma 4.3 the approximation factor for the tour we compute in the original instance will be $(1+\varepsilon)$.

For each separator considered by the algorithm, we guess the number of crossings and for each crossing the length and the number of vertices visited. So the total number of guesses for each crossing is $O(\frac{n^2}{\varepsilon} \cdot n) = O(\frac{n^3}{\varepsilon})$ (because of Lemma 4.3), for a total of $O(\frac{\log n}{\varepsilon} \cdot (\frac{n^3}{\varepsilon})^{O(\log n/\varepsilon)}) = n^{O(\log n/\varepsilon)}$ guesses for a node. If $T(n)$ is the running time of Algorithm 1 when run on a rounded instance of n nodes, then

$$T(n) = n^{O(\log n/\varepsilon)} \cdot 2T\left(\frac{2n}{3}\right),$$

for an overall running time of $n^{O(\log^2 n/\varepsilon)}$.

Notice that the look-up table stores *costs* instead of subtours, so at the end the algorithm has computed the *cost* of a near-optimal tour and not the tour itself. But it is easy to reconstruct this tour from the look-up table and the decision made at each step of the dynamic programming. □

4.2. The Euclidean plane. In section 2, we reduced the minimum latency problem to the problem of finding a covering of the n nodes using $O(\log n/\varepsilon)$ salesman paths. We use a simple modification of Arora's ideas [4] for the Euclidean TSP to show that this set of $O(\log n/\varepsilon)$ salesman paths together have a very simple structure. Thus they can be computed by dynamic programming in quasi-polynomial time in a way similar to the tree case. We will start by describing the algorithm for the Euclidean plane. The generalization to spaces of higher dimension will follow. Our exposition follows the exposition in Arora [4].

4.2.1. The partition. Arora [4] describes a very simple *geometric* partition which we use here to break the given instance into smaller instances. In what follows we assume that all coordinates of the given points are integral. Later we will show

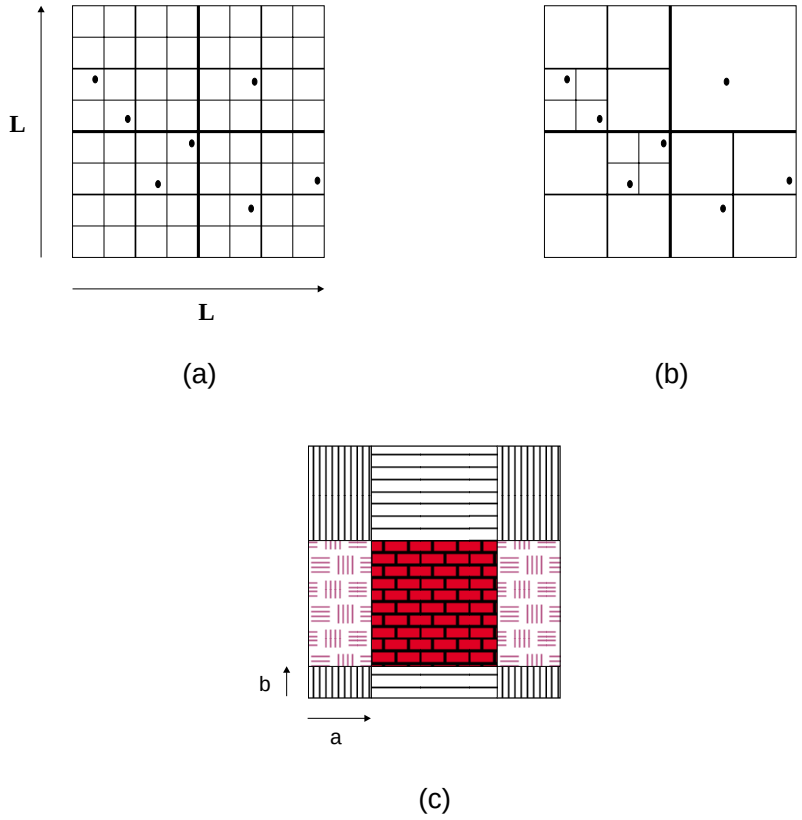


FIG. 4.1. *A dissection* (a)*, the corresponding quadtree* (b)*, and its shift by* (a, b) (c).

how to "perturb" the instance so that this assumption is fulfilled without increasing the cost of our solution by too much.

Suppose that the given instance is bounded by a square, and let L be the size of the smallest axis-aligned bounding square. Then the *dissection* of the bounding square is its recursive partition into smaller squares in the obvious way: break the bounding square into 4 equal squares; then break each smaller square into 4 equal squares and so on until the smallest squares have side size ≤ 1 (and thus cannot contain more than 1 point). This partition defines a tree: the root is the bounding square, and the nodes of the tree at each level are the squares created at the same level of the dissection. Each square has 4 children. Obviously, the tree has $O(L^2)$ nodes, and its depth is $O(\log L)$. If we stop partitioning a square as soon as it contains at most 1 node, then this truncated version of the dissection tree is called a *quadtree*. Examples of a dissection and a quadtree are shown in Figure 4.1.

The partition we are going to use is a randomized version of the quadtree. Imagine

that we pick randomly two integers a, b in $[0, L)$ and then *shift* the dissection defined above along the x- and y-axis by a and b, respectively. At the same time we "wrap-around" this shifted dissection as in Figure 4.1(c). Thus every horizontal line with initial y-coordinate y_1 will now have a new y-coordinate $(y_1 + b) \mod L$, and every vertical line with initial x-coordinate x_1 will now have a new x-coordinate $(x_1 + a) \mod L$. We call this dissection a *randomly shifted dissection*, and the corresponding quadtree (i.e., the quadtree resulting by cutting of the partitioning at squares that contain at most 1 point) is called a *randomly shifted quadtree*. The random (a, b)-shift is crucial for the algorithm, but since we still treat each "wrapped-around" square as a single region the reader can think of the quadtree as the unshifted one in much of what follows.

The geometric partition we just defined resembles in many ways the recursive partition in the tree case. But while a tour on a tree could cross partition boundaries only through a tree node, the boundaries of the quadtree partition are lines that can be crossed at any of an infinite number of points. In what follows we will prove that there is a near optimal tour that crosses not only the boundary of each square in the quadtree only a few times but also that these crossings happen at a small set of prespecified points called *portals*. Each square has 4 portals on its corners and m other equally spaced portals on each side, where m is a power of 2. A portal of a square is a portal in every descendent of the square in the quadtree.

DEFINITION 2. *Let m, k be positive integers. An* (m, k)-light tour *is one that crosses each quadtree boundary at most k times and always at one of its m portals.*

In what follows we prove the existence of an (m, k)-light tour (for appropriate values of m, k) that has near-optimal latency. Then we will describe a dynamic programming algorithm that can compute such a tour in quasi-polynomial time.

4.2.2. The structure theorem for the Euclidean plane. In section 4.2.1 we assumed that the given points have integral coordinates. We will also demand that all internode distances are integers between 8 and $O(n^2)$. Such an instance is called *well-rounded*. Obviously, the given instance need not be well-rounded, but by adding a *rounding step* similar to the one used in the tree case the algorithm can transform the given instance into a well-rounded one without increasing the MLT by too much. We prove the following structure theorem for the Euclidean plane.

THEOREM 4.5 (structure theorem for Euclidean plane). *There exist constants e, f such that the following is true for every integer $n > 0$ and every $\varepsilon > 0$: For every well-rounded Euclidean instance with n nodes, a randomly shifted dissection has with probability at least $1/2$ an associated tour that is $(e \log n/\varepsilon, f \log n/\varepsilon)$-light and whose latency is at most $(1 + \varepsilon)OPT$, where OPT denotes the latency of the MLT.*

Proof. The main ideas are the same as in Arora's proof [4] after we break up the MLT into $O(\log n/\varepsilon)$ salesman paths. The crucial observation is that Arora's proof relies on an expectation calculation that can be extended to the MLT case due to the linearity of expectation.

Let $\mathcal{T}$ be the optimum tour. By applying our main idea, we can break it up into $k = O(\log n/\delta)$ segments, where the ith segment has n_i nodes. We replace each segment by the optimum salesman path for that segment to get a new tour of total latency within a factor $(1+\delta)$ of the optimum. We use the following structure theorem by Arora [4] (Theorem 2 in [4]).

THEOREM 4.6 (Arora [4]). *Let $c > 0$ be any constant. Let the minimum nonzero*

internode distance in a TSP instance be 8*, and let* L *be the size of its bounding box.*[1] *Let shifts* $0 \le a, b \le L$ *be picked randomly. Then with probability at least* $1/2$*, there is a salesman path of cost at most* $(1 + 1/c)TSP$ *that is* (m, r)*-light with respect to the dissection with shift* (a, b)*, where* $m = O(c \log L)$*,* $r = O(c)$*, and* TSP *is the length of the optimal salesman path.*

We apply Theorem 4.6 simultaneously to our $k = O(\log n/\delta)$ salesman paths $T_1, T_2, \ldots, T_k$. The new tour crosses the boundary of each quadtree square at most $O(ck)$ times (since a salesman path never needs to cross a square boundary more than $r = O(c)$ times).

In order to prove Theorem 4.6, Arora [4] proves that the expected increase of a salesman path length after it is modified to become (m, r)-light is bounded as follows:

$$E_{a,b}[\text{increase of salesman path length}] \le \frac{g}{r} TSP, \tag{4.1}$$

where g is a constant.

For the effect of the modified tour on the latency, note that we are interested in a *weighted* sum of salesman path lengths, where the weight assigned to the ith salesman path is $n_i + n_{>i}$ (these are the vertices whose latency is affected by the change in length of T_i). Inequality (4.1) implies that the expected latency increase due to the length increase of path T_i is

$$\begin{aligned} E_{a,b}[\text{total latency increase due to } T_i] &\le \frac{g}{r}(n_i + n_{>i})TSP_i \\ &\overset{(2.1)}{\le} \frac{g(1+\delta)}{r} n_{>i} TSP_i, \end{aligned} \tag{4.2}$$

where TSP_i is the length of the optimum (i.e., shortest) T_i. By linearity of expectation, the total expected latency increase is

$$\begin{aligned} E_{a,b}[\text{total latency increase due to } T_1, \ldots, T_k] &\le \frac{g(1+\delta)}{r} \sum_{i=1}^{k} n_{>i} TSP_i \\ &\le \frac{g(1+\delta)}{r} OPT \end{aligned}$$

because of lower bound (2.2) in section 2.

By picking $\delta = O(\varepsilon)$ and $c = O(1/\varepsilon)$ the total latency increase is no more than $\varepsilon \cdot OPT$, and the new tour satisfies the theorem requirements. □

Thus in this section we proved that there is a tour with latency within a factor $1 + \varepsilon$ of the optimal that crosses each square boundary in the quadtree only $O(\frac{\log n}{\varepsilon})$ times and always through a portal. In the next section we show how to use dynamic programming in order to compute this tour.

4.2.3. The algorithm for the plane. In this section we describe our QPTAS for the Euclidean plane.

Until now we assumed that the given instance is *well-rounded*. Namely, it satisfies two conditions: (i) all nodes have integral coordinates, and (ii) all internode distances are between 1 and $O(n^2/\varepsilon)$. This assumption is easily met if the first step of the algorithm is a **rounding procedure** very similar to the rounding procedure used in the tree case:

[1] In our case $L = O(n^2)$ due to the instance rounding.

1. Let L be the length of the side of the bounding box. Place a grid of granularity $g = \Theta(\delta L/n^2)$, where δ is a constant $(0 < \delta < \varepsilon)$.
2. Move each point to its nearest gridpoint.
3. Divide all edge weights by g.

By an analysis similar to that in the proof of Lemma 4.3 we get the following.

LEMMA 4.7. *The procedure above transforms a given instance of optimal latency OPT into a well-rounded instance of optimal latency increased by at most δ OPT.*

The well-rounded instance has one more property that we will use in the analysis of the algorithm. Namely, the set of possible lengths for parts of the tour is *discrete* with at most $n^{O(1)}$ elements and can be generated quite easily in polynomial time. Hence when we refer to "guesses" for subtour lengths we will refer to values from this polynomially large set.

The algorithm takes a well-rounded instance as its input.

ALGORITHM 2.

1. Build a randomly shifted quadtree for the given instance. Since $L = O(n)$ the quadtree depth is $O(\log n)$ and the number of its nodes is $O(n \log n)$ since it has only n leaves.

   Do the following in a "bottom-up" fashion starting from the leaves of the quadtree:
2. Identify a node (i.e., square) at the current level of the quadtree.
3. "Guess" how many times the tour enters the square, and for each of these times the portals it crosses to enter and leave the square, what is the length of the tour portion after each crossing, and how many points are on that tour portion.
4. Search the dynamic programming look-up table for subtours consistent with the "guesses."
5. Combine the subtours found to create a new bigger subtour. If the new subtour is consistent with the fact that we are looking for a $(O(\log n/\varepsilon), O(\log n/\varepsilon))$-light tour, store this new subtour in the look-up table and go to step 2. □

Again, by "guessing" we refer to exhaustive enumeration of all possible values of the guessed quantities. This enumeration will produce possibly more than one candidate tour. We pick the tour with the minimum total latency as our solution, and this will be the output of Algorithm 2.

At a first glance the algorithm seems to miss a key ingredient for the calculation of a tour from its subtours: the order in which the portals are visited in each square. In fact this piece of information is implicit, since the guessed number of nodes visited after each portal crossing also tells us the order in which they occur in the tour (the first portal visited is the one with the longest subsequent tour length guessed, the second is the one with the second longest subsequent tour length, and so on). Now it is easy to see that at every step of the algorithm we have the information necessary to reconstruct a bigger subtour from the subtours already calculated. As in the tree case, the exhaustive enumeration guarantees that one of the tours constructed is the tour of the structure theorem, Theorem 4.5, so Algorithm 2 is indeed an approximation scheme.

It remains to prove that Algorithm 2 is a QPTAS.

THEOREM 4.8. *Algorithm 2 runs in time $n^{O(\frac{\log n}{\varepsilon})}$ and computes a tour whose latency is at most $(1+\varepsilon)$ OPT.*

Proof. If δ is the constant in the rounding procedure, then we apply Theorem 4.5

so that the approximation factor for the rounded instance is $1+\varepsilon-\delta$. Because of Lemma 4.7 the approximation factor for the tour we compute in the original instance will be $(1+\varepsilon)$.

The running time of the algorithm is dominated by the size of the dynamic programming look-up table. We prove the time bound by induction on the depth of the quadtree. The first level (leaves) contains squares with at most one point in them. The tour we are computing is $(c\log n/\varepsilon, f\log n/\varepsilon)$-light (with c, f being the constants in Theorem 4.5), so a leaf square is entered and left by the tour $O(\log n/\varepsilon)$ times through a pair of portals. This involves enumerating all choices for (a) a multiset of $O(\log n/\varepsilon)$ portals on the four sides of the square and (b) the order in which the portals in (a) are crossed by the (m,r)-light tour, where $m = O(\log n/\varepsilon)$ and $r = O(\log n/\varepsilon)$. It is easy to see that the number of choices in (a) is at most $m^{O(r)} = O(\log n/\varepsilon)^{O(\log n/\varepsilon)}$ and the number of choices in (b) is $r^{O(r)} = O(\log n/\varepsilon)^{O(\log n/\varepsilon)}$, for a total of $2^{O(\log n \log\log n/\varepsilon)}$ choices. In addition we need to "guess" the length of the tour portion after each crossing ($O(n^{O(1)}/\varepsilon)$ possibilities since all distances between points form a set of polynomially many values) and how many points are on that tour portion (n possibilities) for each one of the $O(\log n/\varepsilon)$ possible crossings of the tour through the square boundaries. Notice that since we guessed the number of nodes visited after each portal crossing, we know exactly the tour portion that contains the point in the current square. The total number of guesses is thus no more than

$$\begin{aligned}
&\#\text{ portal arrangements} \times (\text{length} \times \#\text{ points})^{\#\text{ crossings}} \\
&= O\left(2^{\frac{\log n \log\log n}{\varepsilon}}\right) \times \left(O\left(\frac{n^{O(1)}}{\varepsilon}\right) \times n\right)^{O(\frac{\log n}{\varepsilon})} \\
&= n^{O(\frac{\log n}{\varepsilon})}.
\end{aligned}$$

The analysis is the same for the inductive step. Assume that we have calculated all possible subtours for all squares in depth $> i$. Let S be a square at depth i and S_1, S_2, S_3, S_4 be its four children in the quadtree. We guess the same numbers as before for each S_j, $j = 1, \ldots, 4$ ($n^{O(\frac{\log n}{\varepsilon})}$ choices), and we look them up in the look-up table constructed thus far. If the choices we find there are consistent with our current guess, we store this guess in the look-up table and continue. Otherwise the algorithm will reject it and will go on to the next possible guess. So the total extra amount of work for each square in the quadtree is $n^{O(\frac{\log n}{\varepsilon})}$, and the total running time of the algorithm is $O(n\log n) \times n^{O(\frac{\log n}{\varepsilon})} = n^{O(\frac{\log n}{\varepsilon})}$ (recall that the quadtree contains only $O(n\log n)$ nodes). □

Derandomization: The algorithm described is a *randomized* one. But it can be easily derandomized: random shifts a and b take discrete values between 1 and L. By trying all possible $O(n^2)$ values for the pair (a,b) and running the algorithm for each one, we are bound to try one of the "good" values (i.e., one that will give the tour guaranteed by Theorem 4.5). This procedure will increase the running time by a factor $O(n^2)$.

4.3. Euclidean spaces of higher dimension. Since the results in Arora [4] generalize to Euclidean spaces of any *constant* dimension d in a straightforward manner, the plane algorithm for the MLT generalizes to higher dimensions as well. We will give just the changes needed in order for the proofs in the plane case to apply here.

The grid we place in the d-dimensional *cube* of side length L that surrounds the instance to transform it into a well-rounded one has granularity $\Theta(\varepsilon L/(n^2\sqrt{d}))$. Then all internode distances are bound by $O(\sqrt{d}n^2/\varepsilon)$.

Instead of randomly shifted quadtrees we use an obvious extension, the 2^d-ary trees. Instead of squares, we are dealing with d-dimensional cubes, so their boundaries are $(d-1)$-dimensional cubes. The m portals placed on these cubes form an orthogonal *lattice* with granularity $W/m^{\frac{1}{d-1}}$, where W is the side length of the cube we place the portals on. The structure theorem now states the following.

THEOREM 4.9 (structure theorem for Euclidean space of dimension d). *There exist constants e, f such that the following is true for every integer $n > 0$ and every $\varepsilon > 0$. For every well-rounded instance in Euclidean space of dimension d with n nodes, a randomly shifted dissection has with probability at least $1/2$ an associated tour that is $(e \cdot (O(\sqrt{d}\log n/\varepsilon))^d, f \cdot (O(\sqrt{d}/\varepsilon))^d \log n)$-light and whose latency is at most $(1+\varepsilon)OPT$, where OPT denotes the latency of the MLT.*

The dynamic programming algorithm will run in time $O((\frac{\log n}{\varepsilon})^{(\sqrt{d}/\varepsilon)^{O(d)}})n^{O(\frac{\log n}{\varepsilon})}$.

4.4. Extension to other norms. Like Arora's algorithm for the Euclidean TSP, our algorithm for the MLT generalizes to any *Minkowski* norm in $\mathbb{R}^d$. Any symmetric body C that is symmetric around the origin can be used to define a Minkowski norm: the length of $x \in \mathbb{R}^d$ is defined to be $\frac{\|x\|^2}{\|y\|^2}$, where y is the intersection of the surface of C with the line connecting x to the origin. This definition generalizes the l_p norm for $p \geq 1$ (in the case of l_p norm C is the l_p-unit ball centered at the origin).

Distances in any Minkowski norm are within a constant factor of the distances under the l_2 norm. Hence the algorithm described for the Euclidean case works for *any* Minkowski norm as well (with the necessary adjustments of the constants in the structure theorem, Theorem 4.9 and the running time calculations).

4.5. Planar graphs. Our techniques apply also to the MLT problem on *weighted planar graphs* (planar graphs with nonnegative edge lengths). The distance between two nodes is defined as the length of the shortest path in the graph that connects them.

Arora et al. [5] have extended the ideas from Arora [4] to devise a PTAS for the TSP on planar graphs. The first step in their algorithm is the extraction of a *spanner* of the input graph. Informally, a spanner is the subgraph of the input graph that approximates within a factor $1+\varepsilon$ the distances in the original graph but with a total edge length that is much smaller (just $O(1/\varepsilon)$ times the cost of the minimum spanning tree in the input graph). They use a construction by Althofer et al. [2] that runs in polynomial time.

Let G be the input graph and G' its spanner. In order to apply dynamic programming we need the notion of a separator (like the $\frac{1}{3} : \frac{2}{3}$-partition in the tree case or the quadtree in the Euclidean case) and a *hierarchical decomposition* of the instance. The separator in this case is a *Jordan curve* that cuts a "hole" in the graph, thus separating it into an *exterior* (i.e., the hole) and an *exterior*. By defining the *weight* of a planar graph in an appropriate way, Arora et al. [5] show that in polynomial time they can compute such a curve with two essential properties: (a) the interior and exterior weights are a constant fraction of the total weight, and (b) the interior is connected to the exterior only via a set of $k = O(\log n/\varepsilon^2)$ vertices. They show that there is a salesman path whose length is at most $(1+\varepsilon)OPT$ and crosses the boundary of each region at most $O(\log n/\varepsilon^2)$ times. Hence a QPTAS similar to the Euclidean one works for planar graphs, too. The role of the square boundaries is played by the

Jordan curves, and the role of portals is played by the k connecting vertices. Our "guesses" are exactly the same as in the Euclidean case.

THEOREM 4.10. *The algorithm for weighted planar graphs runs in time* $n^{O(\frac{\log n}{\varepsilon^2})}$ *and computes a tour whose latency is at most* $(1+\varepsilon)$ OPT.

Proof. The proof is analogous to the proof of Theorem 4.8. Notice that the approximation scheme for the planar TSP is *not* randomized; therefore the proof of Theorem 4.8 is applied without the expectations. □

4.6. General metrics. In this section we will use our main idea to derive an algorithm for the MLT for general metrics. The only requirement for the distances between points in the given instance is to satisfy the triangle inequalities. For this general case we give a simple *polynomial time* 11.656-approximation algorithm that uses an approximation algorithm for the k-minimum spanning tree (k-MST) as a subroutine. The k-MST problem is defined as the problem of finding the minimum cost tree that spans *exactly* k vertices of a given weighted graph.

Our algorithm has the same flavor as the approximation algorithms in Blum et al. [10] and Goemans and Kleinberg [12]. We note that Goemans and Kleinberg [12] give an improved 7.18-approximation algorithm, which is more complicated.

The general approach is motivated by the observation in section 2, and we use the numbers $n_1, n_2, \ldots, n_k$ defined there, where $k = O(\log n/\varepsilon)$. We will choose $\varepsilon = \sqrt{2}$. Our algorithm for general metrics is quite different to the algorithms for the tree or the Euclidean space, and in this case we have to take into account the fact that due to n and our choice for ε some nodes may be "broken" in two, with each fraction being in a different segment. Let R_i be the fraction of the first node of segment i that belongs to segment i (that means that the rest $1 - Ri$ belongs to segment $i-1$). Note that since segment $i-1$ may end in a whole node (i.e., $\sum_{l=1}^{i-1} n_l$ is an integer), R_i may also take the value 0. R_{k+1} is always 0.

The algorithm is as follows. Let P be the starting node. For $i = 1, \ldots, k$, find, using the algorithm for k-MST, a tour L_i that starts at P and visits $n_i + R_{i+1} - R_i$ nodes that are not visited by $L_1, \ldots, L_{i-1}$. This number of nodes is *integer* and is equal to the number of nodes in segment i, minus the fraction of the start node for segment i that belongs to i (this fraction was visited by L_{i-1}), plus the fraction R_{i+1} of the last node of i that *does not* belong to i but is visited nevertheless by L_i. Of the two possible directions of each tour, pick the one that minimizes the latency. Output the concatenation of these tours (using shortcuts to avoid multiple visits to already visited points). Assume for simplicity that we know the last segment in the tour, which contains $[1/\sqrt{2}] = 1$ vertex and 1 edge.

THEOREM 4.11. *The algorithm just described runs in polynomial time and achieves an approximation factor of* 11.656.

Proof. We analyze this algorithm as follows. Let $\mathcal{T}$ be the optimum tour. Denote by T_i the length of the ith segment of $\mathcal{T}$ (which contains n_i nodes). Then, as in (2.2), a lower bound on OPT, the latency of the optimum tour, is

$$OPT \geq \sum_{i=1}^{k} n_{>i} \cdot T_i, \tag{4.3}$$

where $n_{>i} = n_{i+1} + n_{i+2} + \cdots + n_k$ is the number of nodes appearing in the last $k-i$ segments.

Let d_i be the length of the tour L_i computed by our algorithm. We claim the following.

CLAIM 1.

$$d_i \le (2+\delta) \times 2(T_1 + \cdots + T_i) \text{ for } i = 1, 2, \ldots, k-1. \tag{4.4}$$

Proof of Claim 1. The reason for the "$(2+\delta)$" is that it is the approximation ratio of the k-MST algorithm. From now on we will treat this factor as equal to 2 (by making δ very small, e.g., $\delta = 10^{-6}$). The rest of the expression is explained as follows. Consider the closed tour starting from P, including the first i segments of $\mathcal{T}$, and returning to P. Its length is at most $2(T_1 + T_2 + \cdots + T_i)$, and it includes at least $(n_1 + n_2 + \cdots + n_i + R_{i+1})$ nodes (this is an integer number). At least $n_i + R_{i+1} - R_i$ of these nodes are not in $T_1, \ldots, T_{i-1}$ (since these first $i-1$ segments contain in total only $n_1 + n_2 + \cdots + n_{i-1} + R_i$ nodes). Thus $2(T_1 + T_2 + \cdots + T_i)$ is an upper bound on the length of shortest tour that starts and finishes at P and includes at least $n_i + R_{i+1} - R_i$ nodes not on $T_1, \ldots, T_{i-1}$. This finishes the justification for Claim 1. □

Now notice that the latency of the $n_i + R_{i+1} - R_i$ new nodes visited during tour L_i is upper bounded by $\frac{1}{2}(n_i + R_{i+1} - R_i)d_i + (n_i + R_{i+1} - R_i)\sum_{l=1}^{i-1} d_l$. The second term is an upper bound for the total latency incurred due to the $i-1$ segments of $\mathcal{T}$ preceding the ith segment. The first term is an upper bound for the latency due to the ith segment itself, since the latency in the forward direction plus the latency in the backward direction is equal to $(n_i + R_{i+1} - R_i)d_i$, and we traverse tour L_i in the direction that minimizes the latency of the $n_i + R_{i+1} - R_i$ new points visited. So the total latency of our solution—ignoring the last segment—is upper bounded by

$$\begin{aligned}
A &= \sum_{i=1}^{k-1} \left[(n_i + R_{i+1} - R_i) \left(\frac{1}{2} d_i + \sum_{l=1}^{i-1} d_l \right) \right] \\
&= \frac{1}{2} \sum_{i=1}^{k-1} (n_i + R_{i+1} - R_i) d_i + \sum_{i=1}^{k-1} \sum_{l=1}^{i-1} (n_i + R_{i+1} - R_i) d_l \\
&= \frac{1}{2} \sum_{i=1}^{k-1} (n_i + R_{i+1} - R_i) d_i + \sum_{i=1}^{k-1} d_i (n_{>i} - R_{i+1}) \\
&= \frac{1}{2} \sum_{i=1}^{k-1} n_i d_i + \sum_{i=1}^{k-1} n_{>i} d_i - \frac{1}{2} \sum_{i=1}^{k-1} (R_i + R_{i+1}) d_i \\
&\le \frac{1}{2} \sum_{i=1}^{k-1} n_i d_i + \sum_{i=1}^{k-1} n_{>i} d_i.
\end{aligned}$$

From section 2, $n_i = (1+\varepsilon) n_{i+1}$ and $n_{>i} = \frac{n_i}{\varepsilon}$, so

$$\begin{aligned}
A &\le \frac{1+\varepsilon}{2} \sum_{i=1}^{k-1} n_{i+1} d_i + \frac{1+\varepsilon}{\varepsilon} \sum_{i=1}^{k-1} n_{i+1} d_i \\
&= \frac{(1+\varepsilon)(2+\varepsilon)}{2\varepsilon} \sum_{i=1}^{k-1} n_{i+1} d_i.
\end{aligned}$$

Observing that $n_{>i} = \sum_{j=i+1}^{k} n_j$ and ignoring the last segment (which was com-

puted optimally) we also have

$$\begin{aligned}\sum_{l=1}^{k-1} n_{l+1} d_l &\overset{(4.4)}{\leq} 4 \sum_{l=1}^{k-1} n_{l+1} \sum_{i=1}^{l} T_i \\ &= 4 \sum_{l=1}^{k-1} n_{>l} T_l \\ &\leq 4 \times OPT,\end{aligned}$$

and thus

$$A \leq 2 \frac{(1+\varepsilon)(2+\varepsilon)}{\varepsilon}\, OPT.$$

The factor $\frac{(1+\varepsilon)(2+\varepsilon)}{\varepsilon}$ is minimized for $\varepsilon = \sqrt{2}$, and its value is $3+2\sqrt{2} = 5.828$. Thus $A \leq 11.565\ OPT$. □

5. Extending the MLT problem. Our main idea is general enough to apply to a broader category of objective functions. In this section we study the following *weighted* version of MLT.

WEIGHTED MINIMUM LATENCY TOUR (WMLT)

Input: A set of n points (one of them designated as the starting point p_1), a symmetric distance matrix $[d_{ij}]$, and an integral weight $w_i \geq 0$ for each node $i = 1, 2, \ldots, n$.

Output: A tour $p_1 \to p_2 \to \cdots \to p_n$ that visits *all* points and minimizes the total *weighted* latency

$$\sum_{i=2}^{n} \left[w_{p_i} \sum_{j=1}^{i-1} d_{p_j, p_{j+1}} \right]. \tag{5.1}$$

Obviously, this is an extension to the MLT problem: the contribution of each node to the objective function is the length of the tour from the start to the first visit at the node, *weighted by the weight of the node* w_i. When $w_i = 1$, for all i we get the MLT problem as defined in section 1. Notice that again we can express the objective function in a form similar to formula (1.2), namely,

$$\sum_{i=1}^{n-1} \left[\sum_{j=i+1}^{n} w_{p_j} \right] d_{p_i, p_{i+1}}. \tag{5.2}$$

We study instances where d defines a metric (i.e., distances satisfy the triangle inequalities). When a weight w_i is 0, point i can be moved to the end of a tour without increase to the value of the objective function, and it is enough to solve the problem for the subset of points with nonzero weights. Hence from now on the w_i's will be integers greater than 0.

Let $\mathcal{T} = p_1 \to p_2 \to \cdots \to p_n$ be an optimal tour with total weighted latency OPT. Let $W = \sum_{i=1}^{n} w_{p_i}$ be the total point weight and $\varepsilon > 0$ any parameter. We will assume that the following is true for this instance of WMLT.

ASSUMPTION 1. *$\mathcal{T}$ can be broken into k segments so that in segment i it visits n_i points with total weight W_i given by the following expressions:*

$$W_1 = \frac{\varepsilon\, W}{1+\varepsilon}, \qquad W_i = \frac{W_{i-1}}{1+\varepsilon}, \quad i = 2, 3, \ldots, k. \tag{5.3}$$

Obviously, $k = O(\frac{\log W}{\varepsilon})$. Let the length of the ith segment be T_i. If we let $W_{>i}$ denote the total weight of nodes visited in segments numbered $i+1$ and later, then a simple calculation shows that (and this was the reason for our choice of W_i's)

$$W_{>i} = \sum_{j>i} W_j = \frac{W_i}{\varepsilon} \text{ for every } i = 1, \ldots, k-1. \tag{5.4}$$

As in the case of the MLT, we are going to show that if one replaces each segment by the minimum-cost traveling salesman path on the same subset of nodes, while maintaining the starting and ending points, the new total weighted latency is at most $(1+\varepsilon)OPT$. The lower bound on the OPT is similar to the lower bound in the MLT case:

$$OPT \geq \sum_{j=1}^{m-1} W_{>j} \cdot T_j. \tag{5.5}$$

By replacing each segment T_i with the optimum salesman path T_i' we increase the weighted latency of points within the segment by at most $W_i T_i'$. Hence the total weighted latency of the new tour $\mathcal{T}'$ can be upper bounded as follows:

$$\begin{aligned} \text{cost}(\mathcal{T}') &\leq \sum_{i=1}^{k} W_i T_i' + \sum_{i=1}^{k} W_{>i} T_i' \\ &\leq \varepsilon \cdot \sum_{i=1}^{k} W_{>i} T_i + \sum_{i=1}^{k} W_{>i} T_i \\ &\overset{(5.5)}{\leq} (1+\varepsilon)\ OPT. \end{aligned}$$

Thus we have shown that under Assumption 1, the analogue of Theorem 2.1 can be proven.

THEOREM 5.1. *Let OPT be the total latency of the WMLT under Assumption 1. There exists a tour that is a concatenation of $O(\frac{\log W}{\varepsilon})$ optimal salesman paths and whose total weighted latency is at most $(1+\varepsilon)OPT$.*

Even if, instead of the optimum salesman path in each segment, we use a $(1+\gamma)$-approximate salesman path, then the latency of the tour of Theorem 5.1 is $(1+\gamma\cdot\varepsilon+\gamma+\varepsilon)OPT$.

The approximation-preserving reduction from the MLT to the weighted vehicle routing with per-mile costs in section 3 works also for the WMLT. If the vehicle routing oracle computes a ρ-approximation in polynomial time, our reduction will lead to a $\rho(1+\varepsilon)$-approximate solution for WMLT in $n^{O(\log W/\varepsilon)}$ time. This follows from Theorem 5.1, since it suffices to go over all possible sequences $p_1, p_2, \ldots, p_k$ for $k = O(\log W/\varepsilon)$, and this increases the running time by a factor of at most $O(n^k) = n^{O(\frac{\log W}{\varepsilon})}$.

5.1. Approximation algorithms for the WMLT. First we point out the differences between our rounding procedures for the MLT and the rounding for the WMLT. In this case it suffices to assume w.l.o.g. that the minimum nonzero internode distance is 1 and maximum internode distance is $O(n \cdot W/\varepsilon)$, where $W = \sum_{i=1}^{n} w_i$ and $\varepsilon > 0$ is any parameter. The reason is that if L denotes the diameter of the space, then L is a lower bound on the weighted minimum latency. Again we merge all pairs of nodes with internode distance at most $\frac{\varepsilon L}{n \cdot W}$. This affects the latency of the optimum tour by at most $\varepsilon L \leq \varepsilon OPT$. Furthermore, the ratio of the maximum internode distance and the minimum nonzero internode distance is $O(nW/\varepsilon)$. Since ε is constant, we will often think of the maximum internode distance as $O(nW)$. Note that in the weighted case the internode distance (and therefore the running time of our algorithms) depend directly on W. If W is, for example, exponential in the size of the input, our algorithms will run in exponential time. We give the running times of algorithms described for the MLT when they are applied in the case of WMLT. Their correctness is ensured by Assumption 1 and the extension of our main idea.

THEOREM 5.2. *Algorithm* 1 *runs in time* $(nW)^{O(\log n \log W/\varepsilon)}$ *and computes a tour whose weighted latency is at most* $(1+\varepsilon)\ OPT$.

The proof is identical to Theorem 4.4 and uses Theorem 5.1.

THEOREM 5.3. *The algorithm for the MLT in Euclidean spaces of constant dimension d runs in time* $O((\frac{\log W}{\varepsilon})^{(\sqrt{d}/\varepsilon)^{O(d)}})(nW)^{O(\frac{\log W}{\varepsilon})}$ *and computes a tour whose weighted latency is at most* $(1+\varepsilon)\ OPT$.

For the proof, see section 4.3 and adapt structure theorem, Theorem 4.9 to work with Theorem 5.1. This result extends also to any Minkowski norm (cf. section 4.4).

THEOREM 5.4. *The algorithm for weighted planar graphs runs in time* $(nW)^{\frac{\log W}{\varepsilon}}$ *and computes a tour whose weighted latency is at most* $(1+\varepsilon)\ OPT$.

The proof is identical to Theorem 4.10 in section 4.5.

Acknowledgments. We thank Satish Rao for pointing us to a simplification of our original algorithm and Christos Papadimitriou for bringing [14] to our attention. Sanjeev Arora thanks Naveen Garg and Stefano Leonardi for introducing him to the minimum latency problem.

REFERENCES

[1] F. AFRATI, S. COSMADAKIS, C. PAPADIMITRIOU, G. PAPAGEORGIOU, AND N. PAPAKOSTANTINOU, *The complexity of the traveling repairman problem*, RAIRO Inform. Théor. Appl., 20 (1986), pp. 79–87.

[2] I. ALTHOFER, G. DAS, D. DOBKIN, D. JOSEPH, AND J. SOARES, *On sparse spanners of weighted graphs*, Discrete Comput. Geom., 9 (1993), pp. 81–100.

[3] A. ARCHER AND D. WILLIAMSON, *Faster approximation algorithms for the minimum latency problem*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, pp. 88–96.

[4] S. ARORA, *Polynomial-time approximation schemes for Euclidean TSP and other geometric problems*, J. ACM, 45 (1998), pp. 1–30. Preliminary version in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, Burlington, VT, 1996, pp. 2–12.

[5] S. ARORA, M. GRIGNI, D. KARGER, P. KLEIN, AND A. WOLOSZYN, *A polynomial-time approximation scheme for weighted planar graph TSP*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, San Franciso, CA, 1998, pp. 33–41.

[6] S. ARORA AND G. KARAKOSTAS, *Approximation schemes for minimum latency problems*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1999, pp. 688–693.

[7] S. ARORA AND G. KARAKOSTAS, *A* $2+\varepsilon$ *approximation algorithm for the k-MST problem*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, pp. 754–759.

[8] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and the hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[9] L. Bianco, A. Mingossi, and S. Ricciardelli, *The traveling salesman problem with comulative costs*, Networks, 23 (1993), pp. 81–91.

[10] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, *The minimum latency problem*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, Montreal, QC, Canada, 1994, pp. 163–171.

[11] J. Fakcharoenphol, C. Harrelson, and S. Rao, *The k-traveling repairman problem*, in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, pp. 655–664.

[12] M. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, in Proceedings of the Seventh ACM-SIAM Symposium on Discrete Algorithms, Atlanta, GA, 1996, pp. 152–158.

[13] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.

[14] E. Koutsoupias, C. Papadimitriou, and M. Yannakakis, *Searching a fixed graph*, in Automata, Languages and Programming, Lecture Notes in Comput. Sci. 1099, Springer-Verlag, Berlin, 1996, pp. 280–289.

[15] A. Lucena, *Time-dependent traveling salesman problem—the deliveryman case*, Networks, 20 (1990), pp. 753–763.

[16] E. Minieka, *The delivery man problem on a tree network*, Ann. Oper. Res., 18 (1989), pp. 261–266.

[17] J.S.B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems*, SIAM J. Comput., 28 (1999), pp. 1298–1309.

[18] C. Papadimitriou and M. Yannakakis, *The traveling salesman problem with distances one and two*, Math. Oper. Res., 18 (1993), pp. 1–11.

[19] R.A. Sitters, *The minimum latency problem (MLP) is NP-hard for weighted trees*, in Proceedings of the 9th International IPCO Conference, Cambridge, MA, 2002, Lecture Notes in Comput. Sci. 2337, Springer-Verlag, Berlin, 2002, pp. 230–239.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1338–1355

REACHABILITY AND DISTANCE QUERIES VIA 2-HOP LABELS*

EDITH COHEN†, ERAN HALPERIN‡, HAIM KAPLAN‡, AND URI ZWICK‡

Abstract. Reachability and distance queries in graphs are fundamental to numerous applications, ranging from geographic navigation systems to Internet routing. Some of these applications involve huge graphs and yet require fast query answering. We propose a new data structure for representing all distances in a graph. The data structure is *distributed* in the sense that it may be viewed as assigning *labels* to the vertices, such that a query involving vertices u and v may be answered using only the labels of u and v.

Our labels are based on 2-*hop covers* of the shortest paths, or of all paths, in a graph. For shortest paths, such a cover is a collection S of shortest paths such that, for every two vertices u and v, there is a shortest path from u to v that is a concatenation of *two* paths from S. We describe an efficient algorithm for finding an *almost optimal* 2-hop cover of a given collection of paths. Our approach is general and can be applied to directed or undirected graphs, exact or approximate shortest paths, or to reachability queries.

We study the proposed data structure using a combination of theoretical and experimental means. We implemented our algorithm and checked the size of the resulting data structure on several real-life networks from different application areas. Our experiments show that the total size of the labels is typically not much larger than the network itself, and is usually considerably smaller than an explicit representation of the transitive closure of the network.

Key words. shortest-path queries, reachability queries, distance labels, 2-hop labels

AMS subject classifications. 68W40, 68R99

DOI. 10.1137/S0097539702403098

1. Introduction.

We consider the problem of efficiently answering distance or reachability queries in directed or undirected graphs. We focus on scenarios in which the graph/network is given to us explicitly and we are able to do some preprocessing of it. The generated data structure should be fairly compact and should accelerate query answering by as much as possible. Often, the computational resources available for processing queries are weaker than those available during the preprocessing stage. This is the case, for example, in geographic navigation systems, where the preprocessing may be done on a large computer, while queries are answered using a much weaker processor installed in a car.

There are two naive solutions to the problem. The first is to precompute answers to all possible queries, e.g., by solving the all-pairs shortest-paths (APSP) problem or by computing the transitive closure of the network. Each query could then be answered in constant time. This, however, is not a viable option, especially for sparse networks, as the data structure produced for an n-vertex graph would be of size $\Omega(n^2)$. The second solution approach is to do no preprocessing at all and answer each query, e.g., using a single-source shortest-path (SSSP) computation. The space requirements here are minimal, but answering a query on an m-edge graph may take $\Omega(m)$ time.

We present here a new processing scheme, and a corresponding query-answering algorithm, for answering reachability and distance queries. Our preprocessing scheme

*Received by the editors February 25, 2002; accepted for publication (in revised form) May 15, 2003; published electronically August 29, 2003 DATE. A preliminary version of this paper appeared in the Proceedings of the ACM/SIAM Symposium on Discrete Algorithms (SODA), San Francisco, 2002.

http://www.siam.org/journals/sicomp/32-5/40309.html

†AT&T Labs–Research, 180 Park Avenue, Florham Park, NJ 07932 (edith@research.att.com).

‡School of Computer Science, Faculty of Exact Sciences, Tel-Aviv University, Tel Aviv 69978, Israel (heran@post.tau.ac.il, haimk@post.tau.ac.il, zwick@post.tau.ac.il).

generates a data structure of reasonable size, i.e., not much larger than the original network, and typically much smaller than, say, an explicit representation of the transitive closure of the network. Given the precomputed data structure, our query-answering algorithm can answer distance and reachability queries quickly, much faster than is possible without preprocessing.

The data structure generated by our preprocessing algorithm is distributed. We assign to each vertex of the network a *distance* or *reachability* label such that we can later calculate the distance (or reachability relation) between two vertices using only the labels of these two vertices. Distance labels were considered by, among others, Peleg [10], Gavoille et al. [4], and Thorup and Zwick [12]. All these papers, however, consider only *undirected graphs*, and only worst-case results. We are interested mainly in directed graphs, and in the performance of the proposed labeling scheme on networks that occur in practice.

Our labels are based on the concept of 2-*hop covers*. Let $G = (V, E)$ be a (directed or undirected) graph. For every $u, v \in V$, let P_{uv} be a collection of paths from u to v in G. (For example, P_{uv} may be the set of all shortest paths from u to v.) We define a *hop* to be a pair (h, u), where h is a path in G and $u \in V$, the *handle* of the hop, is one of the endpoints of h. We say that a collection of hops H is a 2-hop cover of the collection $P = \cup_{uv} P_{uv}$ if, for every $u, v \in V$, if $P_{uv} \neq \emptyset$, then there is a path $p \in P_{uv}$ and two hops $(h_1, u) \in H$ and $(h_2, v) \in H$ such that p is the concatenation $h_1 h_2$. We show how to obtain a 2-*hop labeling* from such a cover by mapping each hop to an item in the label of its handle node.

The nature of each such item and the interpretation of the labels depend on the specific variant of the problem that we want to solve. Consider, for example, undirected distance queries. Each set of paths P_{uv} contains all shortest paths between u and v. The item in the label $L(u)$ of u that corresponds to a hop $(h = (u = v_0, \ldots, v_k), u)$ is $(w(h), v_k)$, where $w(h)$ is the weight of the path h. The distance between two nodes u and v can then be obtained from their labels by taking the minimum, over all nodes that appear in *both* $L(u)$ and $L(v)$, of the sum of the two weights.

The total size of the labels is thus $|H|$. (We count here the *number* of hops in the cover, not their combined length.) Each distance/reachability query can then be answered in time that is linear in the size of the corresponding labels—therefore, on *average*, in $O(|H|/n)$ time.

We show that finding a 2-hop cover (and thus, a 2-hop labeling) of minimum size is an NP-hard problem. We present, however, an efficient and practical algorithm, which we implemented, for obtaining almost-optimal 2-hop covers. The size of the 2-hop cover returned by this algorithm is larger than the minimum possible 2-hop cover by at most a logarithmic factor. In practice, we expect the performance ratio of this algorithm to be much better.

Our algorithm produces an almost optimal 2-hop labeling of *any* input graph. Some graphs may have shorter labelings that are not 2-hop labelings. But, for many interesting families of graphs such as planar graphs, the optimal 2-hop labelings are almost as short as the optimal labeling. Thus, without being specifically designed for any such graph family, our algorithm produces almost optimal labelings. It is also expected, as verified by our experiments, to work well on graphs that are, say, "mostly" planar, or "mostly" tree-like, as many practical problem are.

We *conjecture* that any n-vertex, m-edge directed graph has a 2-hop cover, and thus a 2-hop labeling, of total size $\tilde{O}(nm^{1/2})$. We show there exist graphs for which

any distance or reachability labeling is of size $\Omega(nm^{1/2})$.

A useful property of our approach is that by properly selecting the underlying set of paths, labels can be produced for different variants of the shortest paths problem: reachability or distances, directed or undirected, exact or approximate distances, and for the complete or partial set of vertex-pairs. Even though exact directed distance labeling for all pairs of vertices is the most general variant in the sense that all other variants can be reduced to it, the distinction is important, as a less-constrained variant often enjoys a more compact labeling. One such example is the family of directed planar graphs: Reachability or $(1+\epsilon)$-approximate shortest paths have worst-case 2-hop labels of size $O(n \log^2 n)$ (using some slight modification of a construction by Thorup [11]), whereas for exact-distances there is a known $\Omega(n^{4/3})$ lower bound for any labeling scheme [4]. It is also not too hard to construct specific graphs which admit more compact labeling for less-constrained variants. Thus, 2-hop labels should always be produced for the least-constrained appropriate variant.

We have made an experimental study of our proposed labeling scheme. We used several real-world networks and obtained promising results, showing that the size of the labels produced is typically significantly smaller than what would have been required for explicit representation.

2. Reachability and distance queries. Let $G = (V, E)$ be a weighted graph (which may be directed or undirected) with $n = |V|$ and $m = |E|$. If $(u, v) \in E$ is an edge, we let $w(u, v)$ be the weight (or cost, or length) attached to that edge. In networks used in applications, the weights attached to the edges are nonnegative. Our approach, however, works also when there are edges of negative weight. We let $\delta(u, v)$ be the *distance* from u to v in the graph, i.e., the smallest weight of a path from u to v, if one exists, in the graph, or ∞ otherwise. (We assume that there are no negative weight cycles in the graph, and so all distances are well defined.)

We would like to preprocess the graph G and obtain a compact representation of it such that, given a pair of vertices $u, v \in E$, we could quickly answer the following queries:

reach(u, v)**:** Is there a path from u to v in the graph?

dist(u, v)**:** What is the distance from u to v in the graph? Sometimes, we would be satisfied with $(1+\epsilon)$-approximate distances.

first-edge(u, v)**:** Which edge emanating from u is a first edge on a (shortest) path from u to v?

path(u, v)**:** Find a (shortest) path from u to v in the graph.

Reachability queries are of course special cases of (directed) distance queries, as there is a path from u to v if and only if the distance from u to v is finite. If we are interested only in reachability properties, we may assume that the weight of all the edges of the graph is 0. Path queries could be answered using repeated first-edge queries. In some cases, it would be possible to accelerate the processing of repeated first-edge queries so that if a shortest path from u to v contains, say, ℓ edges, then the processing time of the ℓ first-edge queries producing it would require substantially less time than ℓ times the time required by a typical first-edge query.

3. 2-hop labels. A possible scheme for achieving the objective set forth above is the following: During the preprocessing stage, we attach to each vertex u of the graph a relatively short label $L(u)$ such that, for any two vertices u and v, the two labels $L(u)$ and $L(v)$ would contain enough information to answer the required queries. More formally we use the following.

DEFINITION 3.1 (distance labelings). *A* distance labeling *of a weighted, directed or undirected, graph $G = (V, E)$ is a pair (L, F), where $L : V \to \{0,1\}^*$ and $F : \{0,1\}^* \times \{0,1\}^* \to (\mathbb{R} \cup \{\infty\}) \times (E \cup \{\phi\})$, such that for every $u, v \in V$, if $F(L(u), L(v)) = (d, e)$, then $d = \delta(u, v)$, the distance from u to v in the graph. If $d = \infty$, then $e = \phi$. Otherwise, e is the first edge on a (shortest) path from u to v in the graph. The* total bit-size *of the labeling is $\sum_{v \in V} |L(v)|$, where $|L(v)|$ is the length of $L(v)$. The* maximum label size *is naturally $\max_{v \in V} |L(v)|$. We say that a labeling scheme has* linear complexity *if $F(L(v), L(u))$ can be computed in $O(|L(u)| + |L(v)|)$ time.*

Reachability labelings are defined similarly, and $(1+\epsilon)$-approximate distance labelings are defined by requiring $\delta(u, v) \le d \le (1+\epsilon)\delta(u, v)$. We focus on reachability/distance labels of the following form.

DEFINITION 3.2 (2-hop distance labeling). *Let $G = (V, E)$ be a weighted directed graph. A* 2-hop distance labeling *of G assigns to each vertex $v \in V$ a label $L(v) = (L_{in}(v), L_{out}(v))$ such that $L_{in}(v)$ is a collection of pairs $(x, \delta(x, v))$, where $x \in V$, and similarly, $L_{out}(v)$ is a collection of pairs $(x, \delta(v, x))$, where $x \in V$. With a slight abuse of notation, we also consider $L_{in}(v)$ and $L_{out}(v)$ to be subsets of V, and for any two vertices $u, v \in V$ we require that*

$$\delta(u, v) \;=\; \min_{x \in L_{out}(u) \cap L_{in}(v)} \delta(u, x) + \delta(x, v).$$

The size of the labeling is defined to be $\sum_{v \in V} |L_{in}(v)| + |L_{out}(v)|$.

For approximate distances we relax the requirement to $\min_{x \in L_{out}(u) \cap L_{in}(v)} \delta(u, x) + \delta(x, v) \le (1+\epsilon)\delta(u, v)$.

For undirected graphs we can simplify the definition: For every vertex v, there is only one collection $L(v)$ of pairs $(x, \delta(x, v))$, such that for every $u, v \in V$ we have $\delta(u, v) = \min_{x \in L(u) \cap L(v)} \delta(u, x) + \delta(x, v)$. The size of the labeling is then $\sum_{v \in V} |L(v)|$.

A slightly more general class of distance labelings is the following.

DEFINITION 3.3 (Steiner 2-hop distance labeling). *A* Steiner 2-hop distance labeling *is a labeling in which $L_{in}(v)$ and $L_{out}(v)$ consist of pairs of the form $(x, d(x, v))$ and $(x, d(v, x))$, respectively, where $x \in X$, $d(x, v), d(v, x) \in \mathbb{R}$, and X is some arbitrary finite set. We require that*

$$\delta(u, v) \;=\; \min_{x \in L_{out}(u) \cap L_{in}(v)} d(u, x) + d(x, v).$$

(A similar definition can be stated for undirected graphs and for approximate distances.)

Note that any 2-hop distance labeling is a Steiner 2-hop labeling, with $X = V$ and $d(x, v) = \delta(x, v)$.

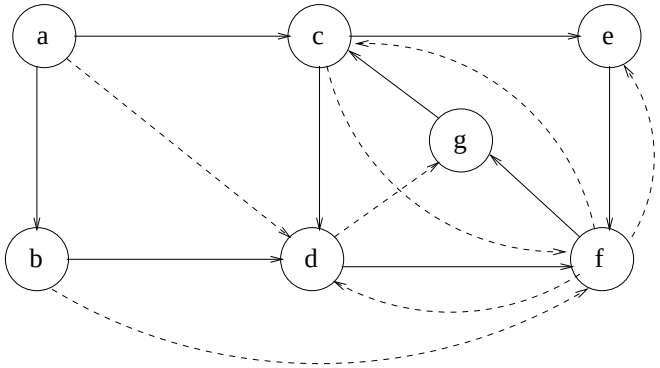
We also use the following definition of 2-hop reachability labels.

DEFINITION 3.4 (2-hop reachability labeling). *Let $G = (V, E)$ be a directed graph. A 2-hop reachability labeling of G assigns to each vertex $v \in V$ a label $L(v) = (L_{in}(v), L_{out}(v))$, such that $L_{in}(v), L_{out}(v) \subseteq V$ and there is a path from every $x \in L_{in}(v)$ to v and from v to every $x \in L_{out}(v)$. Furthermore, for any two vertices $u, v \in V$, we should have*

$$u \rightsquigarrow v \quad \textit{iff} \quad L_{out}(u) \cap L_{in}(v) \neq \phi.$$

The size of the labeling is defined to be $\sum_{v \in V} |L_{in}(v)| + |L_{out}(v)|$.

A *2-hop Steiner reachability labeling* is defined similarly, but we allow $L_{in}(v)$ and $L_{out}(v)$ to contain vertices from an arbitrary finite set X.



v	$L_{in}(v)$	$L_{out}(v)$
a		b, c, d
b		f, d
c	f	d, f
d	f, c	g, f
e	c, f	f
f	d, c	g, c, e
g	d, f	c

FIG. 1. *Solid edges are the edges of the graph. Dashed lines are hops that are not edges of the graph.*

Remarks. (1) In Definition 3.1, we measure the label size in bits. On the other hand, in Definitions 3.2 and 3.4 we measure the size of 2-hop labels by the total number of hops they contain. (2) Our 2-hop labelings, as defined, do not support **first-edge** queries. However, it is easy to add first-edge information to the hops so that they will support such queries.

Given the labels $L(u) = (L_{in}(u), L_{out}(u))$ and $L(v) = (L_{in}(v), L_{out}(v))$, we can easily compute $\delta(u, v)$, the distance from u to v in $O(|L_{out}(u)| + |L_{in}(v)|)$ time. (We keep $L_{out}(u)$ and $L_{in}(v)$ in sorted order and merge them to find their intersection.) We can in fact do it also in $O(\min\{|L_{out}(u)|, |L_{in}(v)|\})$ time, if we keep hash tables for the sets $L_{out}(u)$ and $L_{in}(v)$. As an example for 2-hop reachability labeling, consider the graph in Figure 1 and the 2-hop labeling shown in the table next to it. We assume there that each vertex v is contained, by default, in $L_{in}(v)$ and $L_{out}(v)$.

4. 2-hop covers. Closely related to 2-hop labelings is the notion of a 2-hop cover of a collection of paths in a graph.

DEFINITION 4.1 (2-hop cover). *Let $G = (V, E)$ be a graph. For every $u, v \in V$, let P_{uv} be a collection of paths from u to v (for undirected graphs we have $P_{uv} \equiv P_{vu}$). Let $P = \{P_{uv}\}$. We define a* hop *to be a pair (h, u), where h is a path in G and $u \in V$ is one of the endpoints of h. We refer to u as the* handle *of the hop. A collection of hops H is said to be a 2-hop cover of P if for every $u, v \in V$ such that $P_{uv} \neq \phi$ there is a path $p \in P_{uv}$, and two hops $(h_1, u) \in H$ and $(h_2, v) \in H$, such that $p = h_1 h_2$, i.e., p is the concatenation of h_1 and h_2. The size of the cover is $|H|$, the number of hops in H.*

Intuitively, a path p can be covered by two hops if p is a concatenation of the two hop paths *and* the handles of the hops are the endpoints of p. We obtain 2-hop labels from 2-hop covers by mapping a hop with handle v to an item in the label of the node v. Thus, the size of this labeling is equal to the size of the 2-hop cover. Specifically, we can obtain reachability labels from a 2-hop cover H of the set of *all paths* in G by setting $L_{in}(v) = \{x \mid ((x \rightsquigarrow v), v) \in H\}$ and $L_{out}(v) = \{x \mid ((v \rightsquigarrow x), v) \in H\}$. The converse also holds: From reachability labels of G we can obtain a 2-hop cover of the same size by adding to H a hop consisting of an arbitrary path $v \rightsquigarrow x$ and endpoint v if $x \in L_{out}(v)$ and a hop consisting of an arbitrary path $x \rightsquigarrow v$ and endpoint v if $x \in L_{in}(v)$. Similarly we can obtain 2-hop distance labels from a 2-hop cover of the set of *all shortest paths* in G, and vice versa. We could also obtain approximate

distance labels from a 2-hop cover of the set of all approximate shortest paths and vice versa. Thus we obtain that the size of an optimal 2-hop cover for the set of all paths, the set of all shortest paths, and the set of all approximate shortest paths is equal to the minimum size of a 2-hop reachability labeling, 2-hop distance labeling, and 2-hop approximate distance labelings, respectively.

The common property of the set of all paths, the set of all shortest paths, and the set of all approximate shortest paths, which makes their 2-hop covers correspond to 2-hop labelings, is the following.

DEFINITION 4.2 (hop invariance). *A set of paths $P = \{P_{uv}\}$ in a graph $G = (V, E)$ is said to be* hop invariant *if there exists a collection Q of paths such that*

- *for any two vertices v_i and v_j, there is at most one path $q_{ij} \in Q$ such that $v_i \rightsquigarrow^{q_{ij}} v_j$;*
- *whenever $v_i \rightsquigarrow^{p_1} v_j \rightsquigarrow^{p_2} v_k$ is a path of P, then q_{ij} and q_{jk} exist and $q_{ij}q_{jk} \in P$.*

LEMMA 4.1. *A set P consisting of all shortest paths between a particular set of vertex-pairs (in directed or undirected graphs) is 2-hop invariant. In particular, the set of all shortest paths is hop invariant. A similar statement holds for $(1+\epsilon)$-approximate shortest paths and for the set of all directed paths.*

Proof. In all these cases, it is sufficient to place in Q an arbitrary *shortest* path connecting each pair of nodes. □

THEOREM 4.1. *Finding a minimum 2-hop cover of a collection P of shortest paths in a directed graph is an NP-hard problem.*

Proof. We reduce 3-SAT to the decision version of the minimum 2-hop cover problem of shortest paths in directed graphs. Let S be an instance of 3-SAT with variables $x_1, x_2, x_3, \ldots, x_n$ and clauses $C_1, C_2, \ldots, C_m$. We build an instance for the 2-hop cover problem as follows. The graph G consists of a pivot vertex p together with a collection of variable gadgets and clause gadgets. The variable gadget corresponding to a variable x consists of

1. six vertices a_1^x, a_2^x, b^x, $b^{\overline{x}}$, e_1^x, and e_2^x,
2. eight arcs (a_1^x, b^x), $(a_1^x, b^{\overline{x}})$, (a_2^x, b^x), $(a_2^x, b^{\overline{x}})$, (b^x, e_1^x), $(b^{\overline{x}}, e_1^x)$, (b^x, e_2^x), and $(b^{\overline{x}}, e_2^x)$.

We connect the pivot p to the variable gadget by adding the arcs (p, b^x) and $(p, b^{\overline{x}})$.

Let C be a clause $C = (X, Y, Z)$, where X, Y, and Z are literals of the variables x, y, and z, respectively. For each such clause C we add a vertex c and the arcs (b^X, c), (b^Y, c), and (b^Z, c) (i.e., if $X = x$, we add the arc (b^x, c), and if $X = \overline{x}$, we add the arc $(b^{\overline{x}}, c)$, and similarly for y and z).

For every variable x we define $P_{a_i^x e_j^x}$, $i, j \in \{1, 2\}$, to be the set of directed paths from a_i^x to e_j^x. We also define $P_{pe_j^x}$ to consist of the directed paths from p to e_j^x, $j \in \{1, 2\}$. We call the sets $P_{a_i^x e_j^x}$ and $P_{pe_j^x}$ the *variable paths* that correspond to variable x. For a clause $C = (X, Y, Z)$ we define $P_{b^X c}$, $P_{b^Y c}$, and $P_{b^Z c}$ to be the set containing the directed path of a single edge from b^X to c, from b^Y to c, and from b^Z to c, respectively. We also define P_{pc} to consist of the three directed paths from p to c. We define every P_{uv} not mentioned above to be empty.

We claim that the 3-SAT instance S is satisfiable if and only if the 2-hop cover instance has a solution of size at most $5n + 3m$.

The proof of this claim is as follows. Assume S is satisfiable, and let f be a satisfying assignment. We define a 2-hop cover H as follows. For every variable x if $f(x) = 1$, we add the hops $((a_1^x, b^x), a_1^x)$, $((a_2^x, b^x), a_2^x)$, $((p, b^x), p)$, $((b^x, e_1^x), e_1^x)$, and $((b^x, e_2^x), e_2^x)$ to H. Otherwise we add the hops $((a_1^x, b^{\overline{x}}), a_1^x)$, $((a_2^x, b^{\overline{x}}), a_2^x)$, $((p, b^{\overline{x}}), p)$,

$((b^{\overline{x}}, e_1^x), e_1^x)$, and $((b^{\overline{x}}, e_2^x), e_2^x)$ to H. For each clause $C = (X, Y, Z)$ we add the hops $((b^X, c), c)$, $((b^Y, c), c)$, and $((b^Z, c), c)$ to H. It is straightforward to check that if f is satisfying, then H is indeed a cover. The size of H is $5n + 3m$.

For the other direction, let H be a 2-hop cover of size $5n + 3m$. To cover the sets $P_{b^x c}$ and $P_{b^{\overline{x}} c}$ for all variables x and clauses C, the cover H must include $3m$ distinct hops of the form (b^X, c) for every clause C and literal $X \in C$. To cover the variable paths corresponding to x we need at least five hops, consisting of arcs incident with vertices from the variable gadget corresponding to x, and p.

Since H is a 2-hop cover of size $5n + 3m$, we cover the variable paths of each variable with exactly five hops. It is easy to check that the only way to cover the variable paths with five hops is either to include the hops $((a_1^x, b^x), a_1^x)$, $((a_2^x, b^x), a_2^x)$, $((p, b^x), p)$, $((b^x, e_1^x), e_1^x)$, and $((b^x, e_2^x), e_2^x)$ in H or to include the hops $((a_1^x, b^{\overline{x}}), a_1^x)$, $((a_2^x, b^{\overline{x}}), a_2^x)$, $((p, b^{\overline{x}}), p)$, $((b^{\overline{x}}, e_1^x), e_1^x)$, and $((b^{\overline{x}}, e_2^x), e_2^x)$ in H. We define a satisfying assignment to S by setting $f(x) = 1$ if and only if the former set of hops is in H. Since H also covers the sets P_{pc} for all clauses C, it follows that S must be a satisfying truth assignment. □

We can, however, efficiently find an almost optimal 2-hop cover if the collection of paths P is hop invariant.

THEOREM 4.2. *Let $G = (V, E)$ be a graph with $|V| = n$, and let P be a hop invariant set of paths in G. Then there is an efficient algorithm for finding a 2-hop cover of P whose size is larger than the smallest such cover by at most an $O(\log n)$ factor.*

Before proving the theorem, we introduce some notation. As before, let P_{uv}, for $u, v \in V$, be the paths of P that start at u and end at v. Let $B_{uv} \subseteq V$ be the set of vertices that appear on paths from P_{uv}. (Note that $u, v \in B_{uv}$ if $P_{uv} \neq \phi$.) A moment of reflection shows that the size of the minimum 2-hop cover of P depends only on the sets B_{uv}, for $u, v \in V$. We also need to define the *densest subgraph* problem and state some results that are known for it.

DEFINITION 4.3 (densest subgraph). *The* densest subgraph problem *is defined as follows: Given an undirected graph $G = (V, E)$, find a subset $S \subseteq V$ for which the average degree in the subgraph induced by S is maximized, i.e., a set that maximizes the ratio $|E(S)|/|S|$, where $E(S)$ is the set of edges connecting two vertices of S.*

The densest subgraph problem can be solved exactly in polynomial time using flow techniques. One such algorithm is given by Lawler [8, Chapter 4]. The best time bound currently available for the problem is $O(mn \log(n^2/m))$, due to Gallo, Grigoriadis, and Tarjan [3]. It is obtained by reducing the densest subgraph problem to a parametric min-cut problem and then solving it using a parametric max-flow algorithm whose running time is the same as the running time of the nonparametric max-flow algorithm of Goldberg and Tarjan [5].

Of more practical interest is a much simpler linear time 2-approximation algorithm for the densest subgraph problem, which is a slight modification of an algorithm mentioned by Kortsarz and Peleg [7]. This algorithm iteratively removes a vertex of minimum degree from the graph. This generates a sequence of n subgraphs of the original graph. The algorithm returns the densest of these subgraphs. It is not difficult to check that this algorithm can be implemented to run in linear time, and that it is a 2-approximation algorithm for the densest subgraph problem; i.e., the average degree in the subgraph returned is at least half of the average degree in the densest possible subgraph.

We can now present a proof of Theorem 4.2.

Proof of Theorem 4.2. We cast the problem of finding a minimum 2-hop cover of P as a minimum set cover problem. We then apply the greedy algorithm and find a cover that is larger than the optimal cover by at most a logarithmic factor (Chvátal [1], Johnson [6], Lovász [9]). One difficulty that arises is that the resulting set cover instance is huge. We show, however, that it is possible to apply the greedy algorithm to this set cover instance *without* generating it explicitly.

We first recall the flow of the greedy algorithm of the set cover problem. The instance of the set cover problem is a ground set T, and a set $\mathcal{S}$ of subsets of T. For each $S \in \mathcal{S}$, there is an associated weight $w(S)$. The goal is to find a subset $\mathcal{U} \subseteq \mathcal{S}$ such that $\cup_{S \in \mathcal{U}} S = T$ and $\sum_{S \in \mathcal{U}} w(S_i)$ is minimized. The greedy algorithm for the problem is the following. We maintain the set of uncovered elements T', which is initialized to $T' = T$. In each iteration of the algorithm, we add to $\mathcal{U}$ a set S, which maximizes the ratio $\frac{|S \cap T'|}{w(S)}$. We iterate until $T' = \phi$.

The set cover instance corresponding to the 2-hop cover instance is constructed as follows. The ground set of elements to be covered is $T = \{(u, v) \mid P_{uv} \neq \phi\}$. For each vertex $w \in V$ and two subsets $C_{in}, C_{out} \subseteq V$, we have a set

$$S(C_{in}, w, C_{out}) \;=\; \{\, (u, v) \in T \mid u \in C_{in}\,,\; v \in C_{out}\,,\; w \in B_{uv}\,\}.$$

The weight attached to this set is $|C_{in}| + |C_{out}|$. The vertex w is called the *center* of the set $S(C_{in}, w, C_{out})$. The goal is to find a collection of such sets of minimum total weight that cover T. (Note that the collection of sets is *exponential* in size.)

The proof that this set cover instance is equivalent to the 2-hop cover problem is as follows: If H is a 2-hop directed cover of minimum size, we let $C_{in}(w) = \{u|((u, w), u) \in H\}$ and $C_{out}(w) = \{u|((w, u), u) \in H\}$. We can then cover the set T using the sets $S(C_{in}(w), w, C_{out}(w))$ for $w \in V$.

Conversely, we first claim that we may assume w.l.o.g. that the minimum cover contains at most one set with a given center. If a cover contains two sets with the same center, i.e., $S(C'_{in}, w, C'_{out})$ and $S(C''_{in}, w, C''_{out})$, then these two sets can be replaced, without increasing the size of the cover, by the set $S(C'_{in} \cup C''_{in}, w, C'_{out} \cup C''_{out})$.

Thus, let $S(C_{in}(w), w, C_{out}(w))$ be the set corresponding to $w \in V$. We can now define a corresponding 2-hop cover in the following way: $((u, w), u) \in H$ if and only if $u \in C_{in}(w)$, and $((w, u), u) \in H$ if and only if $u \in C_{out}(w)$.

To see that this is indeed a 2-hop cover, consider a pair (u, v) such that $P_{uv} \neq \emptyset$. From the construction of the set cover instance, there exists $w \in B_{uv}$ such that $u \in C_{in}(w)$ and $v \in C_{out}(w)$, and thus the hops $((u, w), u)$ and $((w, v), v)$ are in H.

We next have to show that we can efficiently apply the greedy algorithm to this exponential size set cover instance. Let T' be the part of T that is still uncovered. Initially $T' \leftarrow T$. In each step of the greedy algorithm we are supposed to find a set S with the best ratio $\frac{|S \cap T'|}{w(S)}$. In the directed case we are looking for a set $S(C_{in}, w, C_{out})$ for which the ratio

$$\frac{|S(C_{in}, w, C_{out}) \cap T'|}{|C_{in}| + |C_{out}|}$$

is maximized.

To do that we find, for every $w \in V$, the set $S(w)$ which maximizes the above ratio over all the sets $S(C_{in}, w, C_{out})$, in which w is their center. We construct an auxiliary undirected bipartite graph $G_w = (V_w, E_w)$, which we call *the center graph of* w, in the following way. The vertex set V_w contains two vertices v_{in} and v_{out} for each vertex v

of the original graph G. We have the undirected edge $(u_{out}, v_{in}) \in E_w$ if and only if $(u, v) \in T'$ and $w \in B_{uv}$. Many of the vertices in G_w may be isolated and can therefore be removed from the graph. It is straightforward to prove that the problem of finding the sets C_{in} and C_{out} that maximize the ratio $|S(C_{in}, w, C_{out}) \cap T'|/(|C_{in}| + |C_{out}|)$ is exactly the problem of finding the *densest subgraph* of G_w.

We solve this problem, computing $S(w)$ for each center $w \in V$, and finally choose the vertex w for which $S(w)$ has the best ratio. We then add the corresponding set $S(w)$ to the cover, update T', and repeat until T' is empty. It is shown by Johnson [6], Lovász [9], and Chvátal [1] that the greedy heuristic achieves a performance ratio of H_t for the set cover problem, where t is the number of elements to be covered and H_t is the Harmonic number. For our problem, the number of elements is equal to the number of vertex-pairs such that there is at least one path in P between them. Thus, we obtain an approximation ratio of

$$H_{|\{ij|P_{ij}\neq\emptyset\}|} \leq H_{n^2} < 2\log n + O(1).$$

This construction is slightly different for undirected paths. The set T to be covered is a set of (unordered) vertex-pairs $\{u, v\}$ such that $B_{uv} \neq \emptyset$. For each vertex $w \in V$ and a subset $C \subseteq V$, we have a set $S(C, w) = \{\{u, v\} \mid u \in C,\ v \in C,\ w \in B_{uv}\}$. The proof that this set cover problem is equivalent to our original 2-hop cover problem is essentially as for the directed case: Let $S(C(w), w)$ be the set corresponding to w; then $((u, w), w) \in H$ if and only if $u \in C(w)$. To solve the set cover instance, we are interested in $S(C, w)$, which maximizes $\frac{|S(C,w)\cap T'|}{|C|}$. We again first solve this maximization problem separately for each w. Here the auxiliary graph $G_w = (V, E_w)$ contains a single copy of each vertex of V and generally is not bipartite. We have the edge $(u, v) \in E_w$ if and only if $(u, v) \in T'$ and $w \in B_{uv}$. Similarly, the problem of finding the set $S(w) = (C, w)$ that maximizes the ratio $|S(C, w) \cap T'|/|C|$ is then exactly the problem of finding the *densest subgraph* of G_w. □

Our approximation algorithm has ingredients similar to those of an approximation algorithm given by Kortsarz and Peleg [7] for the 2-spanner problem.

4.1. A simple example.

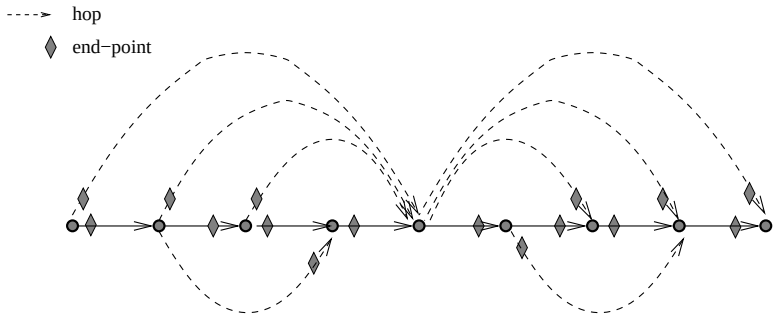


FIG. 2. *A 2-hop cover of a path. Hops are dashed lines or original edges. The diamonds indicate which endpoint is the handle of the hop.*

To better visualize the working of our algorithm, we demonstrate its operation for the special case where the input is a graph that consists of single directed path (see Figure 2). We name the nodes on the path by consecutive integers $(1, 2, 3, \ldots, n)$. Consider the center graph G_j of vertex j. For every i, k such that $i < j$ and $k > j$

there is an edge from i_{out} to k_{in} in G_j that corresponds to the path from i to k going through j. The subgraph induced by the $x = j-1$ hops of the form $l \rightsquigarrow j$ $(l < j)$ and the $y = n-j$ hops of the form $j \rightsquigarrow l$ $(l > j)$ has density $\frac{xy}{x+y}$. Since $x+y = n-1$, the density is maximized when $x = y = (n-1)/2$. Thus, the first densest subgraph that our algorithm will pick would be the whole center graph that corresponds to the node $j = \lceil n/2 \rceil$. Following this choice, all shortest paths traversing the center are covered, and we are left with two independent subproblems: the paths induced by the first half of the original path (up to and excluding the middle node) and the paths induced by the second half of the original path. By the same argument, the algorithm will continue and pick the center graph of the middle node in each of the subproblems. It is easy to see that the total number of hops selected by the algorithm is $O(n \log n)$.

Note that if we consider the corresponding 2-hop labels, then the maximum label of a node (number of hops in which the node constitutes the marked endpoint) is $O(\log n)$. Observe that even though the algorithm tends to select large center graphs, the corresponding labels are stored only at the "outer" nodes. Thus, it tends to perform well also according to the maximum label metric.

The algorithm works similarly on trees, iteratively selecting as a center the "center of gravity" of the subtree and thus generating labeling of size $O(n \log n)$ with a maximum label of size $O(\log n)$.

4.2. Specific graph families. We have seen that our algorithm generates 2-hop labeling of size $O(n \log n)$ for all trees (for any variant of the problem). This matches a lower bound of $\Omega(n \log^2 n)$ on the *bit-size* of any tree labeling, due to Gavoille et al. [4].

Some other graph families are known to have compact 2-hop labeling: Graphs with separator-decomposition of size $O(n^\mu)$ have 2-hop labels of size $O(n^{1+\mu} \log n)$ (e.g., see [2]). It is possible that the optimal 2-hop labeling for small-separator graphs is $o(n^{1+\mu} \log n)$. (This would be the case if our Conjecture 5.1 is true.)

For planar graphs, Thorup [11] shows that $O(n \log n)$ size reachability and approximate distance labels are possible. It follows from his construction that there are corresponding 2-hop labels of size $O(n \log^2 n)$. We are not aware of a matching lower bound for 2-hop reachability labelings; thus, it is possible that there are optimal 2-hop reachability labelings of size $O(n \log n)$ for planar graphs. Thorup's result is particularly intriguing since there is an $\Omega(n^{4/3})$ lower bound on exact distance labels for planar graphs.

Another particular family of interest consists of graphs with bounded degree d where for each pair of vertices there is at least one path between them of length at most $D = \log_d n + o(\log_d n)$ edges. For example, a random d-regular graph will have this property with high probability. For such graphs we can obtain 2-hop labelings of size $O(n^{1.5+\epsilon})$ by picking the hops $((v,u),v)$ and $((v,u),u)$ for every two nodes v, u such that there is a path of at most $\lceil D/2 \rceil$ edges from v to u. Observe that the total number of hops is $O(nd^{D/2}) = O(n^{3/2+\epsilon})$.

5. Lower bounds on reachability and distance labelings. In this section we prove lower bounds on the size of reachability labels and 2-hop reachability labels in directed graphs. The bounds we prove also hold for distance labelings in directed and undirected graphs, and it is straightforward to extend the proofs for these cases. We begin with the following simple lemma that gives a lower bound on the size in bits of any reachability labeling (not necessarily a 2-hop reachability labeling).

LEMMA 5.1. *Any reachability labeling scheme must assign some n-vertex m-edge graph reachability labels of total size $\Omega(m \log(n^2/m))$ bits.*

Proof. Consider the set of all directed m-edge graphs on the set of vertices $V = \{1, 2, \ldots, n\}$ in which all edges are directed from $V_1 = \{1, 2, \ldots, n/2\}$ to $V_2 = \{n/2 + 1, n/2 + 2, \ldots, n\}$. (Assume that n is even.) There are $\binom{(n/2)^2}{m}$ such graphs, and each of them has a distinct transitive closure. Hence, no two of these graphs may be assigned reachability labels that are identical for every vertex. It follows that most of these graphs must be assigned reachability labels of total size $\Omega(\log \binom{(n/2)^2}{m}) = \Omega(m \log(n^2/m))$. ∎

Note that each graph from the family of graphs considered in the proof of Lemma 5.1 may be assigned 2-hop reachability labels of total size $O(m \log n)$ bits. We simply let $L_{out}(v) = \{u \in V_2 \mid (v, u) \in E\}$ for every $v \in V_1$, $L_{in}(v) = \{v\}$ for every $v \in V_2$, and all other sets be empty. Thus, 2-hop reachability labels are almost optimal for this family of graphs. The corollary below summarizes this discussion and shows that 2-hop labels are almost optimal in the following sense.

COROLLARY 5.1. *Let $g(n, L)$ be the collection of n-node graphs with 2-hop labels (distance or reachability) of size L ($L \log n$ bits). Then any general labeling scheme assigns labels of maximum size $\Omega(L \log(n^2/m))$ bits on $g(n, L)$.*

Proof. Consider the family of graphs considered in the proof of Lemma 5.1, with n nodes and $L = m$ edges. These graphs have 2-hop labels of size L. The corollary follows from the proof of the lemma. ∎

We next use Lemma 5.1 to obtain a stronger lower bound on the total size of reachability labels.

THEOREM 5.1. *Any reachability labeling scheme must assign to some graphs with n vertices and m edges reachability labels of total size $\Omega(nm^{1/2})$ bits.*

Proof. Consider graphs of the following form. Start with a bipartite graph on vertex sets V_1 and V_2, where $|V_1| = |V_2| = m^{1/2}$, with $m/2$ directed edges going from V_1 to V_2. Next, make each vertex of V_1 the center of a star with $n/m^{1/2}$ leaves. All these leaves are disjoint, and the edges from these leaves are directed towards the vertices of V_1. Similarly, make each vertex of V_2 the center of a star with $n/m^{1/2}$ leaves. Edges this time are directed away from the vertices of V_2. The total number of vertices in this graph is $2m^{1/2} \cdot n/m^{1/2} + 2m^{1/2} = 2(n + m^{1/2}) = \Theta(n)$, and the number of edges is $2m^{1/2} \cdot n/m^{1/2} + m/2 = m/2 + 2n = \Theta(m)$. (Note that $n \le m \le n^2$ and therefore $m^{1/2} \le n$.)

Let U_i, for $1 \le i \le n/m^{1/2}$, be the set composed of the ith leaf of every star. For every i, the reachability relation restricted to U_i is isomorphic to the reachability relation on the set $V_1 \cup V_2$. It follows from Lemma 5.1 that, for at least one such graph, we need reachability labels of total size at least $\Omega(m)$ bits. Thus, the total size of the labels attached to all the vertices must be at least $\Omega(n/m^{1/2} \cdot m) = \Omega(nm^{1/2})$ bits. ∎

Our next lemma will allow us to obtain a slightly better lower bound than the one in Theorem 5.1 on the size of 2-*hop reachability labelings*. The lemma establishes a lower bound on the size of the optimal 2-hop cover for a set of paths. To specify this lemma we need the following definitions. Given a set P of paths and $v, w \in V$, we define $\overline{h}_v(w)$ to be the number of vertices reachable from v via a path in P going through w. We also define $\underline{h}_v(w)$ to be the number of vertices from which you can reach v through a path in P going through w. Formally, $\overline{h}_v(w) = |\{u \mid w \in B_{vu}\}|$ and $\underline{h}_v(w) = |\{u \mid w \in B_{uv}\}|$. The quantity $\overline{h}_v(w)$ (and similarly $\underline{h}_v(w)$) can be thought of as a bound on the effectiveness of selecting the hop $((w, v), v)$ to our 2-hop

cover; $\overline{h}_v(w)$ bounds the maximum number of pairs such that this particular hop can participate in their cover. Similarly, we can bound the maximum effectiveness of the covering of a particular pair by considering the maximum, over all 2-hop combinations that can cover it, of the maximum number of pairs that the less effective of these two hops can participate in covering. Formally, for each pair of nodes (a, b) we define the *efficiency of covering* P_{ab} as follows:

$$\text{eff}(a,b) = \max_{p \in P_{ab}} \max_{v \in p} \min\{\overline{h}_a(v), \underline{h}_b(v)\}.$$

LEMMA 5.2. *For any 2-hop cover,*

$$|H| \geq \sum_{(a,b)|P_{ab} \neq \emptyset} 1/\text{eff}(a,b).$$

Proof. Any pair (a, b) is eventually covered by two hops. One of these hops participates in at most $\text{eff}(a, b)$ paths. Thus, if the cost of each hop is partitioned among the pairs it covers, (a, b)'s share is at least $1/\text{eff}(a, b)$. Thus, the total number of hops is at least $\sum_{ab} 1/\text{eff}(a, b)$. □

We can obtain a slightly stronger lower bound for 2-hop reachability labels than the bound in Theorem 5.1 for general labelings. In Theorem 5.1 the lower bound of $\Omega(nm^{1/2})$ is on the size of the labels in bits, whereas in the following theorem the lower bound is on the total number of hops in the 2-hop labels.

THEOREM 5.2. *There exist n-vertex m-edge graphs for which any 2-hop reachability labeling scheme must have a total size of* $\Omega(nm^{1/2})$.

Proof. The graphs we consider are a subset of those used in the proof of Theorem 5.1. We start with a *complete* bipartite graph with $|V_1| = |V_2| = m^{1/2}$. As before, we make each vertex of V_1 and V_2 the center of a star with $n/m^{1/2}$ leaves. The proof follows using Lemma 5.2. □

Last, we show that some graphs with large 2-hop reachability labels have much shorter Steiner labels.

COROLLARY 5.2. *There exists a family of graphs with Steiner reachability labels of size* $O(n)$*, where the best proper 2-hop reachability labels are of size* $\Omega(nm^{1/2})$.

Proof. The graphs in the proof of Theorem 5.2 can be viewed as 4-layer graphs, with v reachable from u if and only if the layer of u is lower than the layer of v. We use the set $X = \{x_{1,2}, x_{1,3}, x_{1,4}, x_{2,3}, x_{2,4}, x_{3,4}\}$; the identifier $x_{i,j}$ is placed in $L_{out}(v)$ for all v in layer i, and in $L_{in}(v)$ for all v in layer j. □

A similar, slightly more involved construction can be used to generate $O(n)$-size Steiner labels for the undirected uniform-weighted version of this construction as follows.

COROLLARY 5.3. *There exists a family of undirected graphs with Steiner distance labels of size* $O(n)$ *where the best proper 2-hop distance labels are of size* $\Omega(nm^{1/2})$.

We conjecture that the bound given in Theorem 5.2 is best possible.

CONJECTURE 5.1. *Let* $G = (V, E)$ *be a directed graph with* $|V| = n$ *and* $|E| = m$*, and let* P *be the set of all shortest paths in* G*. Then there is a 2-hop cover of* P *of size* $O(nm^{1/2})$.

6. The pairs-cover problem. The algorithm in the proof of Theorem 4.2 applies to the following more general problem.

PROBLEM 6.1 (pairs-cover). **Input:** $P_1, \ldots, P_k$*, where each* P_i *is a set of pairs or triples of integers.*

Output: *A set H of pairs of integers of minimum size such that, for every $i = 1, \ldots, k$, one of the following holds:*

- *there exists a triple $(c, a, b) \in P_i$ such that $(c, a) \in H$ and $(c, b) \in H$, or*
- *there exists a pair $(c, a) \in P_i$ such that $(c, a) \in H$.*

The problem of computing a 2-hop cover for a hop invariant set of paths P can be mapped to the following pairs-cover instance: Let $V = \{v_0, \ldots, v_{n-1}\}$ be the set of nodes of G. Each nonempty $P_{v_i v_j}$ is mapped to a different P_ℓ in the pairs-cover instance.

For each v_c that is an interior node on some path in $P_{v_i v_j}$ we append the following pairs/triples to the corresponding P_ℓ (mapping is slightly different for undirected or directed graphs):

- For directed paths, we include the triple $(c, n + i, j)$ and the pairs (i, j) and $(j, n + i)$ in P_ℓ;
- For undirected paths, we include the triple (c, i, j) and the pairs (i, j) and (j, i) in P_ℓ.

We interpret the output H of the pairs-cover instance as follows: If $(c, i) \in H$ $(i < n)$, then the hop $((v_c, v_i), v_i)$ is placed in the cover. (Recall that since P is hop invariant, hops can be specified by their endpoints.) For undirected instances hops contain undirected paths. For directed instances hops contain directed paths. We can also have a pair $(c, i) \in H$ with $(i \geq n)$, in which case we rewrite it as $(c, n + a) \in H$ (where $i = n + a$). In this case we place the hop $((v_a, v_c), v_a)$ in the cover.

We now discuss another natural set of problems which this framework covers. We first consider a variant of the 2-hop cover problem where hops do not have handles. Thus, every hop is simply a path, and P_{uv} is covered if there are $P \in P_{uv}$ and two paths $h_1, h_2 \in H$ such that $P = h_1 h_2$. Observe that any solution with handles can be converted to a solution without handles by omitting the handles. Conversely, a solution without handles can be converted to a solution with handles by introducing two hops, with both endpoints as handles for every original hop. Thus, the optimal solution to the variant with handles is at most twice that of the variant without handles. Our greedy algorithm can hence be applied to the version without handles, albeit with a loss of another factor of 2 in the approximation ratio. Curiously, it seems that there is no natural modification of the greedy algorithm that obtains the same approximation ratio for the variant without handles.

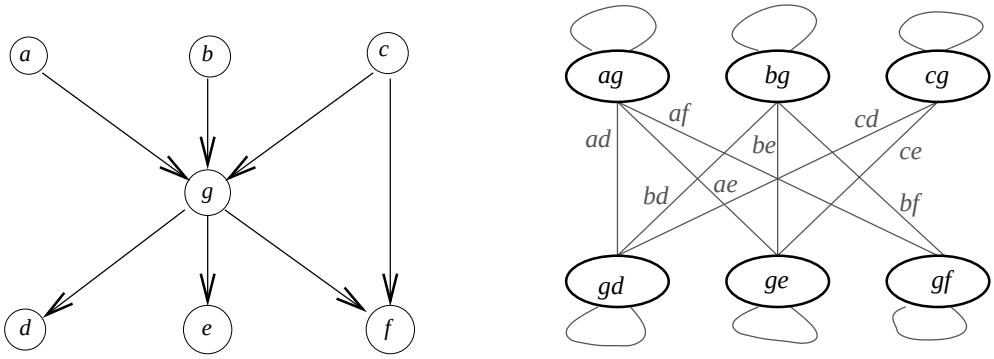
The variant with handles better fits the labeling application, as there is 1-1 correspondence between 2-hop covers with handles and 2-hop labelings, but variants without handles can have applications.

One such conceivable application is the following problem.

PROBLEM 6.2 (minimum 2-hop representation). *Let P be an arbitrary collection of strings. A minimum 2-hop representation of P is a set of substrings H such that every string $s \in P$ is a concatenation $s = h_1 h_2$ of two strings $h_1, h_2 \in H$.*

7. Implementation. Our implementation computes the cover in two phases. The first phase is different for different applications (i.e., reachability, approximate or exact, directed or undirected distances) and produces an instance of the pairs-cover problem. The second phase is an approximation algorithm for the pairs-cover problem.

The second phase then constructs a data structure of *center-graphs*: With each number that appears as the first coordinate in some triple we associate a *center*. For each center c we construct an (undirected) *center-graph*. There is a node i in the center-graph of c if and only if there is a pair (c, i) or a triple of the form $(c, \bullet, i)$ or



Directed Shortest Paths

Center-graph of node "g"

P_{ad}	=	$\{(a, d\text{-in}),\ (d, a\text{-out}),\ (g, a\text{-out}, d\text{-in})\}$
P_{ag}	=	$\{(a, g\text{-in}),\ (g, a\text{-out})\}$
P_{cf}	=	$\{(c, f\text{-in}),\ (f, c\text{-out})\}$
P_{gd}	=	$\{(g, d\text{-in}),\ (d, g\text{-out})\}$

Some entries of the pairs-cover instance

FIG. 3. *This is an example for the construction of a pairs-cover instance and center-graphs for an instance of the directed shortest-paths problem on a graph. The graph has seven nodes labeled* $\{a, b, c, d, e, f, g\}$. *The finite distances in these graphs consist of seven ordered pairs with distance* 1 *and eight ordered pairs of distance* 2. *The corresponding pairs-cover instance has a set of pairs/triples for each of these fifteen ordered pairs of nodes with a finite distance. The table shows some of the entries; other entries are symmetric. Specifically, the entries for* P_{xy} *for all shortest paths of length* 2 *(*$ad, ae, af, bd, be, bf, cd, ce$*) are symmetric to* P_{ad}; P_{bg} *and* P_{cg} *are symmetric to* P_{ag}; *and* P_{ge} *and* P_{gf} *are symmetric to* P_{gd}. *The right-hand figure shows the center-graph for the center* g. *There are six nodes in this center-graph, which correspond to shortest-paths for the ordered pairs* $\{ag, bg, cg, gd, ge, gf\}$. *These hops are denoted by a-out, b-out, c-out, d-in, e-in, and* f*-in in the pairs-cover instance. The edges of the center-graph are ordered pairs for which the two hops constitute a shortest path; equivalently,* xy *labels an edge if the triple* $(g, x\text{-out}, y\text{-in})$ *(or* $(g, y\text{-in}, x\text{-out})$*, which is the same) appears in the pairs-cover instance. Self-loops correspond to paths that consist of a single hop. The densest subgraph of this particular center-graph is obtained by selecting all nodes; this subgraph thus covers fourteen of the shortest paths using six hops. Note that* P_{cf} *(the path from* c *to* f*) is not covered by this center-graph, since there is no shortest-path that traverses the node* g. *This path can be covered via a single hop (a single node with self-loop) in the center-graph of* c *or the center-graph of* f. *For this instance, the minimum* 2*-hop cover consists of seven hops.*

$(c, i, \bullet)$ in the input. For each triple (c, i, j) there is an edge (i, j) in the center-graph of c. For each pair (c, i) there is a self-loop on the node i in the center-graph c.[1] Figure 3 provides an example of the construction of a center-graph.

The algorithm proceeds in iterations, where the basic operation in each such iteration is *subgraph selection*: We choose a center-graph and *select* a subset of the remaining nodes and the corresponding incident remaining edges. The selected nodes are removed. The selected subgraph is in fact an (approximate) densest subgraph in this center-graph. Our implementation of approximate densest subgraph (ADS) is linear in the size (number of edges and nodes) of the center-graph: Nodes are

[1]We remark that center-graphs are bipartite when the original network is directed, but this is generally not true with undirected networks.

maintained sorted by remaining degree (initially using bucket-sort), and each edge is looked at once.

Each input set P_ℓ is "responsible" for many edges that lie in different center-graphs. (Each such edge corresponds to a triple or pair in P_ℓ.) In our data structure all these edges are *linked* (via a doubly linked list). Once one of these edges is *selected*, the set P_ℓ is considered *covered*, and all these linked edges are removed from their respective center-graphs. Thus, subgraph selection in one center-graph may result in removed edges at other center-graphs.

The main loop guides the selection of the center-graph from which a subgraph selection is made. Centers are maintained in a heap. The key of a center c is the ratio of edges to nodes in the most recently computed ADS of the center-graph G_c. The heap is initialized through an ADS computation in each center-graph. The main loop repeats the following until all paths are covered (all center-graphs' edges are removed):

- A center c of maximum key is removed from the heap.
- If edges got removed from the center-graph G_c since the last ADS computation was performed, then the ADS is recomputed, and c is placed in the heap with a new key.
- Otherwise, if the center-graph was untouched, the ADS is selected. If unselected edges remain in G_c, then a new ADS is computed, and c is inserted into the heap with a new key.

For this implementation to work correctly, it is important that a center of (approximately) minimum key is selected at each step. To see this, notice that the key of a center-graph (ratio of edges to nodes in the densest subgraph) can only decrease if edges are removed from the graph.

We implemented a variant of the greedy set cover algorithm slightly different from the one outlined in the proof of Theorem 4.2: After the algorithm selects several subsets which correspond to the same center—$S(C_1, w), S(C_2, w), \ldots, S(C_k, w)$—the ratio attached to a remaining subset $S(C, w)$ is updated to

$$|S(C, w) \cap T'| / \left| C \setminus \bigcup_{i=1}^{k} C_i \right|$$

(instead of $|S(C, w) \cap T'|/|C|$), where T' is the set of uncovered edges in the center-graph G_w. This is achieved by removing from the center-graph nodes that are already selected. (Note that edges are removed only if both endpoints are selected.)

Observe that the numerator of this ratio can decrease as subsets are selected in other centers. The denominator can also decrease but only as a result of a subset's being selected from the same center. Even though the ratio associated with a particular subset may increase, it is easy to see that it can never exceed that of the most recently selected densest subgraph from the same center (if it did, then this subset combined with the recent densest subgraph would yield a denser subgraph, hence a contradiction). Thus, the maximum ratio of the densest subgraph of a center is non-increasing. As a new densest subgraph is computed when a center is placed back in the heap, a center with maximum ratio is still found by our heap implementation.

On our data sets, this variant performed better than the variant where the denominator of the ratio of a subset remains $|C|$. We show that the worst-case approximation ratio is the same as for the other variant, as follows.

LEMMA 7.1. *This variant achieves a worst-case approximation ratio of* $O(\log n)$.

Proof. For every set $S = (C_S, w)$, let L_S be the elements of C_S that were already chosen in previous rounds of the algorithm. Let $S_1, \ldots, S_k$ be the sets used by our algorithm, in that order. Consider an element v that was covered for the first time by a set S. We let $w_v = \frac{|C_S \setminus L_S|}{|S \cap T'|}$. Notice that

$$\sum_{i=1}^{k} w(S_i) = \sum_{i=1}^{k} |C_{S_i}| \geq \sum_{i=1}^{k} |C_{S_i} \setminus L_{S_i}| = \sum_{v \in T} w_v.$$

Therefore, it is sufficient to show that when v is the jth element covered, then $w_v \leq \frac{OPT}{|T|-j}$, where OPT is the value of the optimal solution. Before v was covered, $|T'| \geq |T| - j$. Let $O_1, \ldots, O_{k'}$ be the sets of an optimal cover, in which each two sets have different centers. Clearly, $\sum_{i=1}^{k'} |C_{O_i}| = OPT$. Furthermore, $\sum_{i=1}^{k'} |O_i \cap T'| \geq |T'| \geq |T| - j$. Therefore, there is at least one set O_i such that

$$\frac{|O_i \cap T'|}{|C_{O_i} \setminus L_{O_i}|} \geq \frac{|O_i \cap T'|}{|C_{O_i}|} \geq \frac{|T| - j}{OPT}.$$

By definition of the greedy algorithm, the set S which covers v will also satisfy that $\frac{|S \cap T'|}{|C_S \setminus L_S|} \geq \frac{|T|-j}{OPT}$, and thus $w_v \leq \frac{OPT}{|T|-j}$, and the claim follows. ∎

8. Experiments. We used different synthetic and real networks to evaluate our labeling algorithm. Our selection of data sets was guided by two important applications of our labeling scheme, namely, routing and geographic navigation systems. For geographic navigation, distance queries can tell a user the time or distance to reach a desired destination. First-edge queries can be used to generate turn-by-turn driving directions.

Routing tables for packets traveling in a communication network constitute another application of distance labels. The model is that each packet carries its destination label, and the current router obtains the next-hop and, if desired, the "distance," by considering the label at the current router and the destination label. This is somewhat similar to the way Internet routing is performed now: Each packet carries its destination IP-address. The router looks up the IP-address (longest prefix match) in its local routing table in order to obtain the next-hop. Routing tables, however, are growing in size. IP networks typically have small diameter. For inter-AS graphs,[2] the core of the graph has high expansion, whereas intra-AS networks could be almost planar.

ISP-net is the network of a large-backbone ISP (internet service provider). The nodes and (directed) edges of the network correspond to routers and links. This particular ISP uses OSPF (open shortest path first) routing, and the edge-weights are the OSPF weights of the links.

BGP is the set of (directed) paths advertised by BGP (border gateway protocol) routers.[3] The nodes of the network are all ASs in the Internet reachable from the core. The paths are all advertised paths.[4] We considered only paths of three or more

[2] AS stands for an autonomous system, which is an independent subnetwork of the global network.

[3] This data set was given to us by Andre Broido from CAIDA (University of California, San Diego).

[4] In this case the paths are not necessarily shortest paths and also do not necessarily possess the hop invariance property. (Due to business and other considerations, actual BGP routes do not necessary correspond to shortest paths.) We applied only the second phase of the implementation, looking for a 2-hop representation of *every* path in P.

TABLE 1
Parameters and performance for different networks.

Network	# nodes	# edges	# paths (pairs)	Label size	Compression
ISP-net	229	880	38466	2969	13.0
BGP	9236	–	164037	22454	7.3
Roads	548	686	128491	6567	19.6
Grid-10	100	180	2744	843	3.3
Grid-20	400	760	42852	5759	7.44
Grid-30	900	1740	212819	18667	11.40

TABLE 2
Total label size and the maximum size of the label of a particular node for different networks.

Network	# nodes	Label size	Average label	Maximum label	Maximum in/out list
ISP-net	229	2969	13	29	15
BGP	9236	22454	2.4	145	140
Roads	548	6567	12	28	–
Grid-10	100	843	8.43	15	14
Grid-20	400	5759	14.4	30	30
Grid-30	900	18667	20.7	60	58

hops, as shorter paths can be reconstructed by maintaining edge information at each node.

Roads is the road map of Alpine County, CA (USA), obtained from the TIGER census data [13]. This graph is undirected, with weights corresponding to actual distances.

Grid-k is a synthetic network. The underlying network is a $k \times k$ grid. Edges are directed in a Manhattan fashion, with even and odd row-edges directed in opposite directions and, similarly, even and odd column-edges directed in opposite directions. The edge weights were selected uniformly at random from $[1, 100]$.

Table 1 lists for each network the number of nodes, number of edges, number of pairs of nodes such that there is a path from one to the other, and the total size of labels (number of hops). The compression ratio is the ratio of the number of pairs to the total size of the labels. The compression ratio varies with different graph sizes and structures and was between 3 and 19.6. Generally, we expect better ratios for larger graphs. In particular, our analysis shows that for the planar k-grid graphs the total size of the labels is $O(k^3)$, whereas explicit representation is $\Theta(k^4)$; thus the ratio is at least $\Omega(k)$.

So far we have considered the total size of the labels. Another parameter of interest is the maximum label size of a particular node. This parameter is particularly important for distributed applications. It is also relevant since the actual computation of the distance from the labels is linear in the size of the labels. For directed graphs we also consider the maximum size of the in-list or out-list of a node. The average and maximum label sizes for the different networks are listed in Table 2. Although geared to minimizing the total label size, our algorithm seems to perform well also with respect to the maximum-label metric.

Another interesting question regards the dependence of the label size on the number of covered paths. Our algorithm can be set to stop after any given fraction of the paths is covered (and, actually, it provides the same performance guarantees even in this case). The dependence seemed Zipf-like and similar across networks (e.g., 20%–25% of the hops suffices to cover half the paths).

9. Concluding remarks. We have introduced simple and natural distance and reachability labeling schemes for directed and undirected graphs. Our labelings are derived from 2-hop covers of sets of paths in graphs. We give an efficient algorithm for constructing a 2-hop cover whose size is larger than the smallest 2-hop cover by a factor of at most $O(\log n)$. We *conjecture* that there exists a 2-hop cover of size $\tilde{O}(nm^{1/2})$ of the set of shortest paths in any weighted directed graph with n vertices and m edges. Proving, or disproving, this conjecture is perhaps the most interesting problem left open. We have also demonstrated the effectiveness of our schemes by an experimental analysis using synthetic and real networks from applications such as geographic navigation and Internet routing.

REFERENCES

[1] V. Chvátal, *A greedy heuristic for the set-covering problem*, Math. Oper. Res., 4 (1979), pp. 233–235.

[2] E. Cohen, *Efficient parallel shortest-paths in digraphs with a separator decomposition*, J. Algorithms, 21 (1996), pp. 331–357.

[3] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan, *A fast parametric maximum flow algorithm and applications*, SIAM J. Comput., 18 (1989), pp. 30–55.

[4] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz, *Distance labeling in graphs*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, Washington, DC, 2001, SIAM, Philadelphia, pp. 210–219.

[5] A. V. Goldberg and R. E. Tarjan, *A new approach to the maximum flow problem*, J. ACM, 35 (1988), pp. 921–940.

[6] D. S. Johnson, *Approximation algorithms for combinatorical problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.

[7] G. Kortsarz and D. Peleg, *Generating sparse* 2*-spanners*, J. Algorithms, 17 (1994), pp. 222–236.

[8] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Reinhart, and Winston, New York, 1976.

[9] L. Lovász, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.

[10] D. Peleg, *Proximity-preserving labeling schemes*, J. Graph Theory, 33 (2000), pp. 167–176.

[11] M. Thorup, *Compact oracles for reachability and approximate distances in planar digraphs*, in Proceedings of the 42nd IEEE Annual Symposium on Foundations of Computer Science, Las Vegas, NV, 2001.

[12] M. Thorup and U. Zwick, *Approximate distance oracles*, in Proceedings of the 33th Annual ACM Symposium on Theory of Computing, Crete, Greece, 2001, pp. 183–192.

[13] U.S. Census Bureau, *TIGER (Topologically Integrated Geographic Encoding and Referencing System)*, available online at http://www.census.gov/geo/www/tiger.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1356–1369

ON THE COMPLEXITY OF MATRIX PRODUCT*

RAN RAZ†

Abstract. Our main result is a lower bound of $\Omega(m^2 \log m)$ for the size of any arithmetic circuit for the product of two matrices, over the real or complex numbers, as long as the circuit does not use products with field elements of absolute value larger than 1 (where $m \times m$ is the size of each matrix). That is, our lower bound is superlinear in the number of inputs and is applied for circuits that use addition gates, product gates, and products with field elements of absolute value up to 1.

We also prove size-depth tradeoffs for such circuits: We show that if a circuit, as above, is of depth d, then its size is $\Omega(m^{2+1/O(d)})$.

Key words. lower bounds, circuit complexity, algebraic complexity, arithmetic circuits, bounded coefficients model, singular values

AMS subject classification. 68Q17

DOI. 10.1137/S0097539702402147

1. Introduction. Matrix product is among the most extensively studied computational problems. Surprising upper bounds of $O(m^{2+\alpha})$ (where $\alpha < 1$ and $m \times m$ is the size of each matrix) were obtained by Strassen in [Str] and improved in many other works. The best current upper bound (obtained by Coppersmith and Winograd) achieves $\alpha \approx 0.376$ [CW] (see [Gat] for a survey). The best lower bounds, however, are linear lower bounds of between $2.5 \cdot m^2$ and $3 \cdot m^2$ (depending on the field) for the number of products needed [Bsh, Bla, Shp].

In particular, the following seminal problem is still open: Can matrix product be computed by circuits of size $O(m^2)$, that is, circuits of size linear in the number of inputs? Superlinear lower bounds for matrix product are known only for bounded depth circuits [RS]. Note, however, that Strassen's method, as well as many other methods for matrix product, use circuits of larger depth.

The standard computational model for matrix product is by arithmetic circuits over some field F. The inputs for the circuit are the entries of the two matrices, and the allowed gates are product and addition over F. Products with field elements are also allowed. In this work, we take F to be the field of real numbers (all of our results hold for the complex numbers as well), and we restrict our arithmetic circuit in the following way: The circuit cannot use products with field elements of absolute value larger than 1. We call such a circuit *a bounded coefficients arithmetic circuit*.

We prove that any such circuit for matrix product is of size $\Omega(m^2 \log m)$.

More generally, if we require that the circuit does not use products with field elements of absolute value larger than $c = c(m)$ (for any function $c(m) \geq 1$), we obtain that any such circuit for matrix product is of size $\Omega(m^2 \log_{2c} m)$. This follows from the case $c = c(m) = 1$ by a simple reduction (just by replacing each product with a field element of absolute value smaller than or equal to c by up to $\log_2 c$ additions and one product with a field element of absolute value smaller than or equal to 1). Hence, in the rest of the paper, we concentrate on the case $c = 1$.

*Received by the editors February 5, 2002; accepted for publication (in revised form) July 14, 2003; published electronically August 29, 2003. This research was supported by US-Israel BSF grant 08-00349.

http://www.siam.org/journals/sicomp/32-5/40214.html

†Department of Applied Mathematics, Weizmann Institute, Rehovot 76100, Israel (ranraz@wisdom.weizmann.ac.il).

Besides our main result, we also prove size-depth tradeoffs for bounded coefficients arithmetic circuits for matrix product. We show that any such circuit of depth d is of size $\Omega(m^{2+1/O(d)})$. Note that for general arithmetic circuits only much weaker size-depth tradeoffs are known [RS].

Since a preliminary version of our paper has appeared, Burgisser and Lotz used our methods (together with some new methods) to obtain a similar lower bound of $\Omega(n \log n)$ for any bounded coefficients arithmetic circuit for the product of two polynomials of degree n (as well as for the division of two polynomials with reminder) [BL].

1.1. Previous work. Bounded coefficients arithmetic circuits were suggested and motivated as a natural model for arithmetic computations by Morgenstern [Mor] and by Chazelle [Cha]. Morgenstern and Chazelle observed that many algorithms for arithmetic problems (e.g., the fast Fourier transform algorithm) do not use field elements at all (or use only small field elements). Morgenstern and Chazelle were mainly interested in the case of linear functions and proved lower bounds of $\Omega(n \log n)$ for several such functions (e.g., for the Fourier transform). Note that for general arithmetic circuits no superlinear lower bound is known for any linear function (or any constant degree polynomial).

Several works proved size-depth tradeoffs of $\Omega(n^{1+1/O(d)})$ for bounded coefficients arithmetic circuits [NW, Lok, Pud]. As in [Mor, Cha], the focus of these works was linear functions. As far as we know, no previous result was obtained for the complexity of matrix product (or similar functions) in the bounded coefficients model.

1.2. Organization of the paper. The paper is organized as follows. In section 2, we give some basic definitions. In section 3, we give lower bounds for linear functions. These lower bounds are then used in section 4 to prove our lower bound for matrix product. The proof of the main lemma is deferred to section 5. In section 6, we prove our size-depth tradeoff for matrix product.

2. Preliminaries. As mentioned above, we consider arithmetic circuits over the field of real numbers. All of our results hold for the complex numbers as well.

An arithmetic circuit is a directed acyclic graph as follows: Nodes of in-degree 0 are called inputs and are labelled with input variables. Nodes of out-degree 0 are called outputs. Each edge is labelled with a field element (we think of this element as multiplying the outcome of the edge). Each node other than an input is labelled with either $+$ or $\times$ (in the first case the node is a plus gate and in the second case a product gate).

The computation is done in the following way. An input just computes the value of the variable that labels it. For every noninput node v, if $v_1, \ldots, v_k$ are the nodes that fan into v, then we multiply the result of each v_i with the field element that labels the edge that connects it to v. If v is a plus gate we sum all the results; otherwise, v is a product gate, and we multiply all the results. Obviously, each node in the circuit computes a polynomial in the input variables.

The *size* of a circuit C is defined to be the number of edges in it and is denoted by $\mathrm{Size}(C)$. The *depth* of a circuit C is defined to be the length of the longest directed path between an input and an output in C and is denoted by $\mathrm{Depth}(C)$.

We say that an arithmetic circuit (over the real numbers) is a *bounded coefficients arithmetic circuit* if all field elements labelling the edges of the circuit are of absolute value smaller than or equal to 1.

We say that an arithmetic circuit is *linear* if all gates in it are plus gates (i.e., the circuit contains no product gates). Obviously, the outputs of a linear circuit

are linear functions in the input variables. Let $L_1, \ldots, L_k$ be k linear functions (in the variables $z_1, \ldots, z_n$). It is well known (and easy to prove) that (over any field with characteristic 0) any arithmetic circuit for $L_1, \ldots, L_k$ can be translated into a linear circuit for $L_1, \ldots, L_k$, with only a constant-factor increase in the size and depth of the circuit. In the same way, any bounded coefficients arithmetic circuit for $L_1, \ldots, L_k$ can be translated into a bounded coefficients linear circuit for $L_1, \ldots, L_k$, with only a constant-factor increase in the size and depth of the circuit. We can hence assume w.l.o.g. that linear forms $L_1, \ldots, L_k$ are computed by linear circuits. Given an $n \times n$ matrix H, we say that a linear circuit computes H if it computes the linear functions that correspond to the rows of H; that is, the circuit computes the functions $\sum_{j=1}^{n} H_{i,j} \cdot z_j$, where $z_1, \ldots, z_n$ are the input variables for the circuit.

In this paper, we prove lower bounds on the size of circuits for the product of two $m \times m$ matrices. The input for such a circuit is of size $2m^2$, and it consists of two $m \times m$ matrices X, Y. The output is the matrix $X \cdot Y$. That is, there are m^2 outputs, and the (i,j)th output is $\sum_{k=1}^{m} X_{i,k} \cdot Y_{k,j}$. Each output is a bilinear form in X and Y.

Since the product of two matrices is a bilinear form, it is natural to consider bilinear arithmetic circuits for it. We say that an arithmetic circuit is *bilinear* if each product gate in it computes the product of two linear functions, one in the variables $\{X_{i,j}\}$ and the other in the variables $\{Y_{i,j}\}$. Thus, a bilinear circuit has the following structure. First, there are many plus gates, computing linear forms in X and linear forms in Y. Then there is one level of product gates that compute bilinear forms. Finally, there are many plus gates that eventually compute the outputs.

Obviously, the outputs of a bilinear circuit are bilinear functions in the input variables of X and Y. Let $f_1, \ldots, f_k$ be k bilinear functions (in the variables of X and Y). It is well known (and easy to prove) that (over any field with characteristic 0) any arithmetic circuit for $f_1, \ldots, f_k$ can be translated into a bilinear circuit for $f_1, \ldots, f_k$, with only a constant-factor increase in the size and depth of the circuit. In the same way, any bounded coefficients arithmetic circuit for $f_1, \ldots, f_k$ can be translated into a bounded coefficients bilinear circuit for $f_1, \ldots, f_k$, with only a constant-factor increase in the size and depth of the circuit. We can hence assume w.l.o.g. that bilinear forms $f_1, \ldots, f_k$ are computed by bilinear circuits.

3. Lower bounds for linear functions. In this section, we prove lower bounds for the size of bounded coefficients linear circuits. In all that follows, we assume w.l.o.g.[1] that all gates in the circuit are of fan-in 2.

Lower bounds for the size of bounded coefficients linear circuits were first proved by Morgenstern [Mor]. Morgenstern observed that for any matrix H, the size of any bounded coefficients linear circuit for H is bounded from below by $\log_2 |\mathrm{Det}[H]|$. For our purpose, we will need the following simple generalization of this result (Lemma 3.1).

Let $L_1, \ldots, L_k$ be k linear functions in the variables $z_1, \ldots, z_n$. We think of each L_i as a vector in the vector space R^n. For every $1 \leq r \leq n$, denote by $\mathrm{Vol}_r[L_1, \ldots, L_k]$ the maximal volume spanned by r vectors from $\{L_1, \ldots, L_k\}$ and $n-r$ arbitrary unit vectors (i.e., vectors with L^2-norm equal to 1). That is,

$$\mathbf{Vol_r}[\mathbf{L_1}, \ldots, \mathbf{L_k}] = \mathrm{MAX}_{i_1,\ldots,i_r,e_{r+1},\ldots,e_n} |\mathrm{Det}[L_{i_1}, \ldots, L_{i_r}, e_{r+1}, \ldots, e_n]|,$$

where $e_{r+1}, \ldots, e_n$ are arbitrary unit vectors in R^n. Equivalently,

$$\mathbf{Vol_r}[\mathbf{L_1}, \ldots, \mathbf{L_k}] = \mathrm{MAX}_{i_1,\ldots,i_r} (\mathrm{Det}[AA^T])^{1/2},$$

[1] Recall that the size of a circuit is defined as the number of edges in it and not the number of nodes.

where A is the $n \times r$ matrix $[L_{i_1}, \ldots, L_{i_r}]$, and A^T is the transpose of A.

In the same way, for a matrix H of size $n \times n$, we define for every $1 \leq r \leq n$,

$\mathbf{Vol_r[H]} = \mathrm{Vol}_r[L_1, \ldots, L_n]$,
where $L_1, \ldots, L_n$ are the linear forms corresponding to the rows of H.

LEMMA 3.1. *Let C be a bounded coefficients linear circuit for $L_1, \ldots, L_k$. Then, for every $1 \leq r \leq n$,*

$$\mathrm{Size}(C) \geq \log_2(\mathrm{Vol}_r[L_1, \ldots, L_k]).$$

Proof. Let $s = \mathrm{Size}(C)$. Note that since C is a directed acyclic graph, it induces a partial order on its nodes (a node v is larger than a node u if there exists a directed path from u to v). Let $f_1, \ldots, f_s$ be the linear functions corresponding to all nodes in C and such that the order of $f_1, \ldots, f_s$ agrees with the partial order induced by the circuit C and $f_1 = z_1, \ldots, f_n = z_n$ (where $z_1, \ldots, z_n$ are the input variables for the circuit).

Since the order of $f_1, \ldots, f_s$ agrees with the order of the circuit, for every $i > n$ there exist $i_1, i_2 < i$ and c_1, c_2 of absolute value ≤ 1 such that $f_i = c_1 \cdot f_{i_1} + c_2 \cdot f_{i_2}$. Hence, by the linear property of the determinant, it is easy to verify that

$$\mathrm{Vol}_r[f_1, \ldots, f_i] \leq 2 \cdot \mathrm{Vol}_r[f_1, \ldots, f_{i-1}],$$

and, since $\mathrm{Vol}_r[f_1, \ldots, f_n] = 1$, we have

$$\mathrm{Vol}_r[f_1, \ldots, f_s] \leq 2^{s-n} < 2^s.$$

Since $\{f_1, \ldots, f_s\}$ include the functions $L_1, \ldots, L_k$, we have

$$\mathrm{Vol}_r[L_1, \ldots, L_k] \leq \mathrm{Vol}_r[f_1, \ldots, f_s] < 2^s. \qquad \square$$

For a linear function L in n variables and for a vector space $V \subset \mathrm{R}^n$, denote by $\mathbf{Dist}[\mathbf{L}, \mathbf{V}]$ the L^2-distance between L and V (as before, we think of L as a vector in R^n). For r linear functions, $L_1, \ldots, L_r$, denote by $\mathbf{Span}[\mathbf{L_1}, \ldots, \mathbf{L_r}]$ the vector space in R^n spanned by $L_1, \ldots, L_r$. Let $L_1, \ldots, L_k$ be k linear functions in n variables. For every $1 \leq r \leq n$, denote

$\mathbf{Rig_r}[\mathbf{L_1}, \ldots, \mathbf{L_k}] = \mathrm{MIN}_V \mathrm{MAX}_i(\mathrm{Dist}[L_i, V])$,
where $V \subset \mathrm{R}^n$ is a vector space of dimension r.

In the same way, for a matrix H of size $n \times n$, we define for every $1 \leq r \leq n$,

$\mathbf{Rig_r}[\mathbf{H}] = \mathrm{Rig}_r[L_1, \ldots, L_n]$,
where $L_1, \ldots, L_n$ are the linear forms corresponding to the rows of H.

A notion similar (but not identical) to $\mathrm{Rig}_r[H]$ was defined in [Lok] and was used there to prove size-depth tradeoffs for bounded coefficients arithmetic circuits. Here, we connect $\mathrm{Rig}_r[L_1, \ldots, L_k]$ to $\mathrm{Vol}_r[L_1, \ldots, L_k]$, and hence to the size of the smallest bounded coefficients arithmetic circuit for $L_1, \ldots, L_k$.

LEMMA 3.2. *For every k linear functions $L_1, \ldots, L_k$, and every $1 \leq r \leq n$,*

$$\log_2(\mathrm{Vol}_r[L_1, \ldots, L_k]) \geq r \cdot \log_2(\mathrm{Rig}_r[L_1, \ldots, L_k]).$$

Proof. Assume w.l.o.g. that $r < k$ (otherwise, $\mathrm{Rig}_r[L_1, \ldots, L_k] = 0$). Assume w.l.o.g. that the order of $L_1, \ldots, L_k$ is as follows: L_1 is a function $L \in \{L_1, \ldots, L_k\}$

such that $\mathrm{Vol}_1[L]$ is maximal. L_2 is a function $L \in \{L_2, \ldots, L_k\}$ such that $\mathrm{Vol}_2[L_1, L]$ is maximal, and so on (i.e., for every $1 \leq i \leq k$, we have that L_i is a function $L \in \{L_i, \ldots, L_k\}$ such that $\mathrm{Vol}_i[L_1, \ldots, L_{i-1}, L]$ is maximal).

Denote $\upsilon_1 = \mathrm{Vol}_1[L_1]$, and for every $1 < i \leq k$ denote $\upsilon_i = \mathrm{Vol}_i[L_1, \ldots, L_i]/\mathrm{Vol}_{i-1}[L_1, \ldots, L_{i-1}]$. Then, by our assumption on the order of $L_1, \ldots, L_k$, it is easy to verify that

$$\upsilon_1 \geq \upsilon_2 \geq \cdots \geq \upsilon_k.$$

Therefore,

$$\mathrm{Vol}_r[L_1, \ldots, L_r] = \prod_{i=1}^{r} \upsilon_i \geq (\upsilon_{r+1})^r.$$

On the other hand (again by our assumption on the order of $L_1, \ldots, L_k$),

$$\upsilon_{r+1} = \mathrm{MAX}_i(\mathrm{Dist}[L_i, V]),$$

where $V = \mathrm{Span}[L_1, \ldots, L_r]$, and hence

$$\mathrm{Rig}_r[L_1, \ldots, L_k] \leq \upsilon_{r+1}.$$

Thus,

$$\mathrm{Vol}_r[L_1, \ldots, L_k] \geq \mathrm{Vol}_r[L_1, \ldots, L_r] \geq (\upsilon_{r+1})^r \geq (\mathrm{Rig}_r[L_1, \ldots, L_k])^r. \qquad \square$$

Given a matrix H of size $m \times m$, we can use Lemmas 3.1 and 3.2 to prove lower bounds for bounded coefficients arithmetic circuits for H. For our purpose, we will also need to prove lower bounds for bounded coefficients arithmetic circuits for the tensor product $I \otimes H$ (where I is the identity matrix of size $m \times m$). Recall that $I \otimes H$ is a matrix of size $m^2 \times m^2$ that consists of $m \times m$ blocks of size $m \times m$ each such that the m blocks on the diagonal contain copies of the matrix H and all other blocks contain the zero matrix (of size $m \times m$). We will use the following proposition.

PROPOSITION 3.3. *Let H be an arbitrary matrix of size $m \times m$, and let I be the identity matrix of size $m \times m$. Then, for every $1 \leq r \leq m$,*

$$\log_2(\mathrm{Vol}_{r \cdot m}[I \otimes H]) \geq m \cdot \log_2(\mathrm{Vol}_r[H]).$$

Proof. By the properties of the determinant, for every matrix A (of size $m \times m$),

$$\mathrm{Det}[I \otimes A] = (\mathrm{Det}[A])^m.$$

Hence, by the definition of Vol,

$$\mathrm{Vol}_{r \cdot m}[I \otimes H] \geq (\mathrm{Vol}_r[H])^m. \qquad \square$$

COROLLARY 3.4. *Let H be an arbitrary matrix of size $m \times m$, and let I be the identity matrix of size $m \times m$. Let C be a bounded coefficients linear circuit for $I \otimes H$. Then, for every $1 \leq r \leq m$,*

$$\mathrm{Size}(C) \geq r \cdot m \cdot \log_2(\mathrm{Rig}_r[H]).$$

Proof. By Lemma 3.1, Proposition 3.3, and Lemma 3.2,

$$\mathrm{Size}(C) \geq \log_2(\mathrm{Vol}_{r \cdot m}[I \otimes H]) \geq m \cdot \log_2(\mathrm{Vol}_r[H]) \geq m \cdot r \cdot \log_2(\mathrm{Rig}_r[H]). \qquad \square$$

4. Lower bounds for matrix product. In this section, we prove a lower bound for the size of bounded coefficients arithmetic circuits for matrix product. Our bound is based on the lower bounds given in the previous section and on Lemma 4.1. The proof of Lemma 4.1 is given in the next section. In the following lemma, we assume for simplicity that $m/10$ is integer.

LEMMA 4.1 (main lemma). *Let $L_1, \ldots, L_k$ be k linear functions (over* R*) in the m^2 variables $y_{1,1}, \ldots, y_{m,m}$ (we think of $y_{1,1}, \ldots, y_{m,m}$ as the entries of a matrix of size $m \times m$). Denote $r = m/10$, and assume (for simplicity) that m is large enough (i.e., $m > m_0$ for some global constant m_0). Then there exists a matrix Y of size $m \times m$ (over* R*), such that*

1. *for every $1 \leq i \leq k$,*

$$|L_i(Y_{1,1}, \ldots, Y_{m,m})| \leq \mathrm{Rig}_{r \cdot m}[L_1, \ldots, L_k] \cdot (2 \ln k + 10)^{1/2};$$

2.

$$\mathrm{Rig}_r[Y] \geq \sqrt{m/9}.$$

We will now state and prove our main result.

THEOREM 4.2. *Let C be a bounded coefficients arithmetic circuit (over the real or complex numbers) for the product of two matrices of size $m \times m$. Then*

$$\mathrm{Size}(C) = \Omega(m^2 \log m).$$

Proof. First note that w.l.o.g. we can assume that the circuit is over the real numbers. This is true because any circuit over the complex numbers can be translated into a circuit over the real numbers (and vice versa) with a constant-factor increase in its size. As before, we assume w.l.o.g. that all gates in the circuit are of fan-in 2. Recall also that we can assume w.l.o.g. that the circuit is bilinear. We assume w.l.o.g. that m is large enough (and, in particular, $m > m_0$, where m_0 is the global constant from Lemma 4.1), and we assume for simplicity that $m/10$ is integer. Define

$$r = m/10.$$

Assume, for a contradiction to the statement of the theorem, that

$$\mathrm{Size}(C) < 0.001 \cdot m^2 \log_2 m.$$

Denote by $v_1, \ldots, v_k$ the product gates of the circuit C. Since the circuit is bilinear, each product gate v_i computes the product of two linear functions, one in the variables $\{x_{i,j}\}$ (of the first matrix) and the other in the variables $\{y_{i,j}\}$ (of the second matrix). Denote the first linear function by R_i and the second linear function by L_i. Thus, v_i computes the product of $R_i(x_{1,1}, \ldots, x_{m,m})$ and $L_i(y_{1,1}, \ldots, y_{m,m})$.

Consider the linear functions $L_1, \ldots, L_k$. These functions are computed by a linear circuit of size smaller than $0.001 \cdot m^2 \log_2 m$ (in the input variables $y_{1,1}, \ldots, y_{m,m}$). Hence, by Lemmas 3.1 and 3.2,

$$r \cdot m \cdot \log_2(\mathrm{Rig}_{r \cdot m}[L_1, \ldots, L_k]) < 0.001 \cdot m^2 \log_2 m.$$

That is,

$$\mathrm{Rig}_{r \cdot m}[L_1, \ldots, L_k] < m^{1/100}.$$

Hence, by Lemma 4.1, there exists a matrix Y of size $m \times m$ (over R) such that

1. for every $1 \leq i \leq k$,

$$|L_i(Y_{1,1}, \ldots, Y_{m,m})| \leq m^{1/100} \cdot (2 \ln k + 10)^{1/2} < m^{1/99}$$

(for large enough m);

2.

$$\mathrm{Rig}_r[Y] \geq \sqrt{m/9}.$$

We fix the input variables $y_{1,1}, \ldots, y_{m,m}$ to be the entries $Y_{1,1}, \ldots, Y_{m,m}$. Denote the obtained circuit by C'. Since we fixed $y_{1,1}, \ldots, y_{m,m}$, each product gate v_i in C turned into a product with the field element $L_i(Y)$. The circuit C' is hence a linear arithmetic circuit, and it is not hard to see that it computes the matrix $I \otimes Y$ in the input variables $x_{1,1}, \ldots, x_{m,m}$.

The circuit C' is not a bounded coefficients arithmetic circuit, because the absolute value of each field element $L_i(Y)$ is not bounded by 1. We would like to convert C' into a bounded coefficients arithmetic circuit C''. This could be done by replacing the product with each field element $L_i(Y)$ by up to $\log_2 |L_i(Y)|$ additions plus one product with a field element of absolute value ≤ 1. Note, however, that k may be almost as large as the size of C', and hence this method may increase the size of C' by more than a constant factor. Instead, we will convert C' into C'' by the following two steps: First, replace the product with each field element $L_i(Y)$ by a product with the field element $L_i(Y)/m^{1/99}$ (which is of absolute value ≤ 1) and multiply each output of the circuit by the field element $m^{1/99}$. (Since the original circuit C was bilinear, it is not hard to see that this step does not change the outputs of the circuit, and the circuit still computes $I \otimes Y$.) Then replace each product (of an output) with the field element $m^{1/99}$ by up to $\log_2(m^{1/99})$ additions plus one product with a field element of absolute value ≤ 1. Since the number of outputs is m^2, this increases the size of the circuit by at most $(1/99) \cdot m^2 \log_2 m$.

Thus, the obtained circuit C'' is a bounded coefficients arithmetic circuit that computes the matrix $I \otimes Y$ and such that

$$\mathrm{Size}(C'') < (1/90) \cdot m^2 \log_2 m.$$

However, by Corollary 3.4,

$$\mathrm{Size}(C'') \geq r \cdot m \cdot \log_2(\mathrm{Rig}_r[Y]) \geq (1/20) \cdot m^2 \log_2(m/9),$$

which is a contradiction (for large enough m). □

5. Proof of the main lemma. In this section, we give a proof of Lemma 4.1.

We think of each linear function L in the variables $y_{1,1}, \ldots, y_{m,m}$ also as a vector in $\mathrm{R}^{m \times m}$, and we think of each vector in $\mathrm{R}^{m \times m}$ also as a linear function in the variables $y_{1,1}, \ldots, y_{m,m}$. Assignments to the variables $y_{1,1}, \ldots, y_{m,m}$ are matrices Z of size $m \times m$. We think of each such matrix also as a vector in $\mathrm{R}^{m \times m}$, and we think of each vector in $\mathrm{R}^{m \times m}$ also as a matrix of size $m \times m$. Given a linear function L in the variables $y_{1,1}, \ldots, y_{m,m}$ and an assignment Z to $y_{1,1}, \ldots, y_{m,m}$, the value $L(Z)$ is the value of the function L on the assignment Z. The norm that we use in $\mathrm{R}^{m \times m}$ is the L^2-norm. This norm is used to measure distances between vectors and lengths of vectors in $\mathrm{R}^{m \times m}$. Hence, it is also used to measure distances between matrices and norms of matrices, and distances between linear functions and norms of linear functions. We denote the L^2-norm of a linear function L by $\|L\|$, and we denote the

L^2-norm of a matrix Z by $\|Z\|$ (this is known as the Frobenius norm of the matrix Z).

Denote

$$R = \mathrm{Rig}_{r\cdot m}[L_1, \ldots, L_k].$$

By the definition of Rig, there exists a vector space $V \subset \mathrm{R}^{m\times m}$ of dimension $r \cdot m$ such that for every $1 \le i \le k$,

$$\mathrm{Dist}[L_i, V] \le R.$$

Denote by $V^\perp \subset \mathrm{R}^{m\times m}$ the vector space orthogonal to V. Note that $V^\perp$ is a vector space of dimension $(m-r)\cdot m$. For every $1 \le i \le k$, we can write L_i as

$$L_i = L_i'' + L_i',$$

where $L_i'' \in V$ and $L_i' \in V^\perp$. Since $\|L_i'\| = \mathrm{Dist}[L_i, V]$, we have for every $1 \le i \le k$,

$$\|L_i'\| \le R.$$

Recall that we can think of V and $V^\perp$ also as subspaces of matrices Z of size $m \times m$. Obviously, $V^\perp$ is the vector space of all matrices $Z \in \mathrm{R}^{m\times m}$ such that every $L \in V$ satisfies $L(Z) = 0$. In the same way, V is the vector space of all matrices $Z \in \mathrm{R}^{m\times m}$ such that every $L \in V^\perp$ satisfies $L(Z) = 0$.

Our construction for the matrix Y will be probabilistic. We will define a random matrix Y that will satisfy the requirements of the lemma with high probability. The definition of Y will be in two stages. First, define the matrix W in the following way. Each entry $W_{i,j}$ is defined to be an independently chosen Gaussian random variable with expectation 0 and variance 1 (i.e., $W_{i,j}$ is chosen independently according to the distribution $N(0,1)$). Thus, the entries of the matrix W form a multinormal distribution. We can write the matrix W as

$$W = W'' + W',$$

where $W'' \in V$ and $W' \in V^\perp$. We define

$$Y = W'$$

(i.e., Y is the projection of W on $V^\perp$). We will show that with high probability Y satisfies the requirements of the lemma.

Claim 5.1. *With high probability (say, with probability of at least* 0.98*), for every* $1 \le i \le k$,

$$|L_i(Y)| \le R \cdot (2\ln k + 10)^{1/2}.$$

Proof. Note that $L_i''(Y) = L_i''(W') = 0$ and that $L_i'(W'') = 0$. Hence, for every $1 \le i \le k$,

$$L_i(Y) = L_i'(Y) = L_i'(W).$$

Each $L_i'(W)$ is a weighted sum of independently chosen Gaussian random variables with expectation 0 and variance 1, and hence $L_i'(W)$ is a Gaussian random variable with expectation 0 and variance $\|L_i'\|^2$.

Since $\|L_i'\| \leq R$, the probability of the event $|L_i'(W)| > R \cdot (2 \ln k + 10)^{1/2}$ can be bounded by $2/(e^5 \cdot k)$ (see, for example, [ASE, Appendix A]). Hence, by the union bound, with probability of at least 0.98, for every $1 \leq i \leq k$,

$$|L_i(Y)| = |L_i'(W)| \leq R \cdot (2 \ln k + 10)^{1/2}. \qquad \square$$

Thus, with high probability, the matrix Y satisfies the first requirement of the lemma. To prove the second requirement, we will need the following claim.

CLAIM 5.2. *Assume that m is large enough (i.e., $m > m_0$ for some global constant m_0). With high probability (say, with probability of at least 0.97), for any matrix D of size $m \times m$ and rank r,*

$$\|Y - D\| \geq m/3.$$

Proof. For the proof of the claim, we will use the spectral method developed by Lokam in [Lok]. Lokam proves a similar lemma for the Hadamard matrix (and for a generalized Hadamard matrix). We will use a similar method, plus some additional facts and observations, to prove our claim for the matrix Y.

Let A be a matrix of size $m \times m$ (of real or complex numbers). The ith *singular value*, $\sigma_i(A)$, is defined by

$$\sigma_i(A) = \sqrt{\lambda_i(AA^*)},$$

where A^* is the conjugate transpose of A, and $\lambda_i(AA^*)$ is the ith largest eigenvalue of AA^* (for $1 \leq i \leq m$).

It is well known that for every matrix A (of size $m \times m$), there exist unitary matrices U, V (of size $m \times m$) such that U^*AV is a diagonal matrix with values $\sigma_1(A), \ldots, \sigma_m(A)$ on the diagonal (see, e.g., [GV, sec. 2.3]).

For the proof of the claim we will need the following six facts. The facts are true for any constant $\epsilon > 0$. The global constant m_0 (from the statement of the claim) depends on the actual ϵ chosen (i.e., we assume that $m > m_0(\epsilon)$).

1. With high probability (say, with probability of at least 0.99),

$$\|W\| \geq (1 - \epsilon) \cdot m.$$

Proof. Note that $\|W\|^2$ is the sum of the squares of m^2 standard Gaussian random variables. Hence, $\|W\|^2$ is a random variable with expectation m^2 and variance $2m^2$, and (by the central limit theorem) with very high probability its value is very close to its expectation. In particular, for large enough m, the probability for $\|W\| < (1 - \epsilon) \cdot m$ is smaller than 0.01 (this follows, e.g., by Chernoff bounds; see, e.g., [ASE, Appendix A]).

2. With high probability (say, with probability of at least 0.99),

$$\|W''\| \leq (1 + \epsilon) \cdot \sqrt{r \cdot m}.$$

Proof. Recall that the entries of W form a multinormal distribution. Since a multinormal distribution does not change under unitary transformations and since W'' is the projection of W on V, we can present $\|W''\|^2$ as the sum of the squares of $r \cdot m$ standard Gaussian random variables. Hence, $\|W''\|^2$ is a random variable with expectation rm and variance $2rm$, and (by the central limit theorem) with very high probability its value is very close to its expectation. In particular, for large enough m, the probability for $\|W''\| > (1 + \epsilon) \cdot \sqrt{r \cdot m}$ is smaller than 0.01 (this follows, e.g., by Chernoff bounds; see, e.g., [ASE, Appendix A]).

3. With high probability (say, with probability of at least 0.99),
$$\sigma_1(W) < (2+\epsilon)\cdot\sqrt{m}.$$
Proof. The proof was given in [Gem] (see also [Sil]).
4. For any matrix D of size $m \times m$ and rank r,
$$\sigma_{r+1}(D), \ldots, \sigma_m(D) = 0.$$
Proof. As mentioned above, there exist unitary matrices U, V (of size $m \times m$) such that U^*DV is a diagonal matrix with values $\sigma_1(D), \ldots, \sigma_m(D)$ on the diagonal. Since unitary transformations do not change the rank of a matrix, we conclude that $\sigma_{r+1}(D), \ldots, \sigma_m(D) = 0$.
5. For any matrix A of size $m \times m$,
$$\|A\|^2 = \sigma_1^2(A) + \cdots + \sigma_m^2(A).$$
Proof. As mentioned above, there exist unitary matrices U, V (of size $m \times m$) such that U^*AV is a diagonal matrix with values $\sigma_1(A), \ldots, \sigma_m(A)$ on the diagonal. Since unitary transformations do not change the norm of a matrix, we conclude that $\|A\|^2 = \sigma_1^2(A) + \cdots + \sigma_m^2(A)$.
6. For any two matrices A, B of size $m \times m$,
$$\sum_{i=1}^{m} [\sigma_i(A) - \sigma_i(B)]^2 \le \|A - B\|^2.$$
Proof. This inequality is know as the Hoffman–Wielandt inequality [HW]. For a proof of this version of the inequality, see [GV, sec. 8.3].

We are now ready to complete the proof of the claim. Assume that the above six facts are all true for $\epsilon = 0.01$ (for large enough m, this happens with probability of at least 0.97). Let D be any matrix of size $m \times m$ and rank r. By facts 6 and 4,
$$\|W - D\|^2 \ge \sum_{i=1}^{m} [\sigma_i(W) - \sigma_i(D)]^2 \ge \sum_{i=r+1}^{m} [\sigma_i(W)]^2.$$
By fact 5, fact 3, and fact 1 (and since $\epsilon = 0.01$ and $r = m/10$),
$$\sum_{i=r+1}^{m} [\sigma_i(W)]^2 = \|W\|^2 - \sum_{i=1}^{r} [\sigma_i(W)]^2 \ge \|W\|^2 - 4.0401 \cdot r \cdot m \ge 0.98 \cdot m^2 - 0.40401 \cdot m^2.$$
Hence,
$$\|W - D\|^2 \ge (0.75 \cdot m)^2,$$
and, by the triangle inequality and fact 2,
$$\|Y - D\| \ge \|W - D\| - \|W - Y\| \ge 0.75 \cdot m - 0.32 \cdot m > m/3. \qquad \square$$

Let us now finish the proof of Lemma 4.1. Note that if $\mathrm{Rig}_r[Y] < \sqrt{m/9}$, then (by the definition of Rig) there exists a matrix D of rank r such that all rows of $Y - D$ are of L^2-norm $< \sqrt{m/9}$, and hence $\|Y - D\|^2 < m \cdot m/9$ (in contradiction to Claim 5.2).

Thus, by Claim 5.1 we know that with high probability Y satisfies the first requirement of the lemma, and by Claim 5.2 we know that with high probability Y satisfies the second requirement of the lemma. Altogether, with probability of at least 0.95, the matrix Y satisfies both requirements. $\square$

6. Size-depth tradeoffs. The following lemma is implicit in [Lok].

LEMMA 6.1. *Let C be a bounded coefficients linear circuit of size s and depth d for the linear functions $L_1, \ldots, L_k$ (in n variables). Then, for every $1 \leq r \leq n$,*

$$\mathrm{Rig}_r[L_1, \ldots, L_k] \leq \left(\frac{s}{r}\right)^{2d}.$$

Proof. Let $v_1, \ldots, v_l$ be all nodes in C of in-degree larger than s/r. Obviously, $l \leq r$. Denote by $f_1, \ldots, f_l$ the l linear functions outputted at the nodes $v_1, \ldots, v_l$. Denote by C' the circuit C after removing from it the l nodes $v_1, \ldots, v_l$ and all edges connected to them. Then each L_i can be written as $L_i = L''_i + L'_i$, where L''_i is a linear combination of the functions $f_1, \ldots, f_l$, and L'_i is the ith output of the circuit C'.

Since the maximal in-degree in C' is at most s/r and since C' is of depth d, the L^1-norm of each L'_i is bounded by $(s/r)^d$, and hence its L^2-norm is bounded by $(s/r)^{2d}$. Hence, if we denote $V = \mathrm{Span}[f_1, \ldots, f_l]$, then for every $1 \leq i \leq k$ we have $\mathrm{Dist}[L_i, V] \leq (s/r)^{2d}$. Thus, $\mathrm{Rig}_r[L_1, \ldots, L_k] \leq \mathrm{Rig}_l[L_1, \ldots, L_k] \leq (s/r)^{2d}$. ∎

For our size-depth tradeoff for matrix product, we will also need the following version of Lemma 4.1. Note that for the proof of Theorem 4.2 we could have used Lemma 6.2 rather than Lemma 4.1. We preferred to use Lemma 4.1 because it makes the proof of Theorem 4.2 more intuitive.

LEMMA 6.2. *Let $L_1, \ldots, L_k$ be k linear functions (over R) in the m^2 variables $y_{1,1}, \ldots, y_{m,m}$ (we think of $y_{1,1}, \ldots, y_{m,m}$ as the entries of a matrix of size $m \times m$). Denote $r = m/10$, and assume (for simplicity) that m is large enough (i.e., $m > m_0$ for some global constant m_0). Then there exists a matrix Y of size $m \times m$ (over R) such that*

1. *for every $1 \leq i \leq k$,*

$$|L_i(Y_{1,1}, \ldots, Y_{m,m})| \leq \mathrm{Rig}_{r \cdot m}[L_1, \ldots, L_k] \cdot (2 \ln k + 10)^{1/2};$$

2.

$$\mathrm{Rig}_{r \cdot m}[I \otimes Y] \geq \sqrt{m/9}.$$

Proof. The proof is similar to the proof of Lemma 4.1.

We define R, W, W'', and Y as in the proof of Lemma 4.1. Thus, by Claim 5.1, with high probability the matrix Y satisfies the first requirement of the lemma. To prove the second requirement, we will need the following version of Claim 5.2.

CLAIM 6.1. *Assume that m is large enough (i.e., $m > m_0$ for some global constant m_0). With high probability (say, with probability of at least 0.97), for any matrix D of size $m^2 \times m^2$ and rank $r \cdot m$,*

$$\|(I \otimes Y) - D\| \geq m^{1.5}/3.$$

Proof. The proof is similar to the proof of Claim 5.2. The first three facts (out of the six given in the proof of Claim 5.2) are replaced by the following three facts. As before, the facts are true for any constant $\epsilon > 0$. The global constant m_0 (from the statement of the claim) depends on the actual ϵ chosen (i.e., we assume that $m > m_0(\epsilon)$).

1. With high probability (say, with probability of at least 0.99),

$$\|I \otimes W\| \geq (1 - \epsilon) \cdot m^{1.5}.$$

Proof. The proof is obvious, since $\|I \otimes W\| = m^{0.5} \cdot \|W\|$.

2. With high probability (say, with probability of at least 0.99),

$$\|I \otimes W''\| \leq (1+\epsilon) \cdot r^{0.5} \cdot m.$$

Proof. The proof is obvious, since $\|I \otimes W''\| = m^{0.5} \cdot \|W''\|$.

3. With high probability (say, with probability of at least 0.99),

$$\sigma_1(I \otimes W) < (2+\epsilon) \cdot m^{0.5}.$$

Proof. The proof is obvious, since $\sigma_1(I \otimes W) = \sigma_1(W)$.

The proof of the claim is now completed as before. Assume that the above six facts are all true for $\epsilon = 0.01$ (for large enough m, this happens with probability of at least 0.97). Let D be any matrix of size $m^2 \times m^2$ and rank $r \cdot m$. By facts 6 and 4,

$$\|(I \otimes W) - D\|^2 \geq \sum_{i=1}^{m^2} [\sigma_i(I \otimes W) - \sigma_i(D)]^2 \geq \sum_{i=r\cdot m+1}^{m^2} [\sigma_i(I \otimes W)]^2.$$

By fact 5, fact 3, and fact 1 (and since $\epsilon = 0.01$ and $r = m/10$),

$$\sum_{i=r\cdot m+1}^{m^2} [\sigma_i(I \otimes W)]^2 = \|I \otimes W\|^2 - \sum_{i=1}^{r\cdot m} [\sigma_i(I \otimes W)]^2 \geq 0.98 \cdot m^3 - 0.40401 \cdot m^3.$$

Hence,

$$\|(I \otimes W) - D\|^2 \geq (0.75 \cdot m^{1.5})^2,$$

and, by the triangle inequality and fact 2,

$$\|(I \otimes Y) - D\| \geq \|(I \otimes W) - D\| - \|(I \otimes W) - (I \otimes Y)\| > m^{1.5}/3. \qquad \square$$

Let us now finish the proof of Lemma 6.2. Note that if $\mathrm{Rig}_{r\cdot m}[I \otimes Y] < \sqrt{m/9}$, then (by the definition of Rig) there exists a matrix D of rank $r \cdot m$ such that all rows of $(I \otimes Y) - D$ are of L^2-norm $< \sqrt{m/9}$, and hence $\|(I \otimes Y) - D\|^2 < m^2 \cdot m/9$ (in contradiction to Claim 6.1).

Thus, by Claim 5.1 we know that with high probability Y satisfies the first requirement of the lemma, and by Claim 6.1 we know that with high probability Y satisfies the second requirement of the lemma. Altogether, with probability of at least 0.95, the matrix Y satisfies both requirements. $\square$

We will now state and prove our size-depth tradeoff for matrix product. We did not attempt here to optimize the constant ϵ.

THEOREM 6.3. *Let C be a bounded coefficients arithmetic circuit of depth d (over the real or complex numbers) for the product of two matrices of size $m \times m$. Then, for some global constant $\epsilon > 0$ (say, $\epsilon = 1/20$),*

$$\mathrm{Size}(C) = \Omega(m^{2+\epsilon/d}).$$

Proof. The proof follows along the lines of the proof of Theorem 4.2. As before, w.l.o.g. we assume that the circuit is over the reals and that the circuit is bilinear. We assume w.l.o.g. that m is large enough (and, in particular, $m > m_0$, where m_0 is the global constant from Lemma 6.2), and we assume for simplicity that $m/10$ is integer. Define

$$r = m/10.$$

Assume, for a contradiction to the statement of the lemma, that

$$\text{Size}(C) < 0.001 \cdot m^{2+\epsilon/d}.$$

As before, denote by $v_1, \ldots, v_k$ the product gates of the circuit C. Since the circuit is bilinear, each product gate v_i computes the product of two linear functions, one in the variables $\{x_{i,j}\}$ (of the first matrix) and the other in the variables $\{y_{i,j}\}$ (of the second matrix). Denote the first linear function by R_i and the second linear function by L_i. Thus, v_i computes the product of $R_i(x_{1,1}, \ldots, x_{m,m})$ and $L_i(y_{1,1}, \ldots, y_{m,m})$.

Consider the linear functions $L_1, \ldots, L_k$. These functions are computed by a linear circuit of depth at most d and size at most $0.001 \cdot m^{2+\epsilon/d}$ (in the input variables $y_{1,1}, \ldots, y_{m,m}$). Hence, by Lemma 6.1,

$$\text{Rig}_{r \cdot m}[L_1, \ldots, L_k] < (0.01)^{2d} \cdot m^{2\epsilon}.$$

Hence, by Lemma 6.2, there exists a matrix Y of size $m \times m$ (over R), such that

1. for every $1 \leq i \leq k$,

$$|L_i(Y_{1,1}, \ldots, Y_{m,m})| \leq (0.01)^{2d} \cdot m^{2\epsilon} \cdot (2 \ln k + 10)^{1/2} < (0.01)^{2d} \cdot m^{2\epsilon} \cdot \ln m$$

(for large enough m);

2.

$$\text{Rig}_{r \cdot m}[I \otimes Y] \geq \sqrt{m/9}.$$

Denote

$$c = (0.01)^{2d} \cdot m^{2\epsilon} \cdot \ln m.$$

We fix the input variables $y_{1,1}, \ldots, y_{m,m}$ to be the entries $Y_{1,1}, \ldots, Y_{m,m}$. Denote the obtained circuit by C'. Since we fixed $y_{1,1}, \ldots, y_{m,m}$, each product gate v_i in C turned into a product with the field element $L_i(Y)$. The circuit C' is hence a linear arithmetic circuit for the matrix $I \otimes Y$ in the input variables $x_{1,1}, \ldots, x_{m,m}$.

As before, the circuit C' is not a bounded coefficients arithmetic circuit. We will convert C' into a bounded coefficients arithmetic circuit C'' by the following two steps: First, replace the product with each field element $L_i(Y)$ by a product with the field element $L_i(Y)/c$ (which is of absolute value ≤ 1) and multiply each output of the circuit by the field element c. Then replace each product (of an output) with the field element c by $2d$ consecutive additions of fan-in $c^{1/2d}$ each (plus one product with a field element of absolute value ≤ 1). Since the number of outputs is m^2, this increases the size of the circuit by at most $m^2 \cdot 2d \cdot c^{1/2d}$, and hence the size of C'' is at most $0.01 \cdot m^{2+\epsilon/d} \cdot \ln m$ (for large enough m).

Thus, the obtained circuit C'' is a bounded coefficients arithmetic circuit of depth $3d$ that computes the matrix $I \otimes Y$ and such that

$$\text{Size}(C'') < 0.01 \cdot m^{2+\epsilon/d} \cdot \ln m.$$

However, since C'' is of depth $3d$, by Lemma 6.1,

$$\text{Size}(C'') \geq r \cdot m \cdot (\text{Rig}_{r \cdot m}[I \otimes Y])^{1/6d} \geq 0.01 \cdot m^{2+1/12d},$$

which is a contradiction (for large enough m and, say, $\epsilon = 1/20$). □

Acknowledgments. I would like to thank Pavel Pudlak, Sasha Razborov, Amir Shpilka, and Avi Wigderson for very helpful conversations.

REFERENCES

[ASE] N. Alon, J. H. Spencer, and P. Erdös, *The Probabilistic Method*, John Wiley and Sons, New York, 1992.

[Bla] M. Bläser, *A* $2.5n$ *lower bound for the rank of* $n \times n$ *matrix multiplication over arbitrary fields*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, New York, NY, 1999, pp. 45–50.

[Bsh] N. H. Bshouty, *A lower bound for matrix multiplication*, SIAM J. Comput., 18 (1989), pp. 759–765.

[BL] P. Burgisser and M. Lotz, *Lower bounds on the bounded coefficient complexity of bilinear maps*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, Vancouver, BC, Canada, 2002.

[Cha] B. Chazelle, *A spectral approach to lower bounds with applications to geometric searching*, SIAM J. Comput., 27 (1998), pp. 545–556.

[CW] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[Gat] J. V. Z. Gathen, *Algebraic complexity theory*, Ann. Rev. Comput. Sci., 3 (1988), pp. 317–347.

[Gem] S. Geman, *A limit theorem for the norm of random matrices*, Ann. Probab., 8 (1980), pp. 252–261.

[GV] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[HW] A. J. Hoffman and H. W. Wielandt, *The variation of the spectrum of normal matrices*, Duke Math. J., 20 (1953), pp. 37–39.

[Lok] S. V. Lokam, *Spectral methods for matrix rigidity with applications to size-depth tradeoffs and communication complexity*, in Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, pp. 6–15.

[Mor] J. Morgenstern, *Note on a lower bound of the linear complexity of the fast fourier transform*, J. Assoc. Comput. Mach., 20 (1973), pp. 305–306.

[NW] N. Nissan and A. Wigderson, *On the complexity of bilinear forms*, in Proceedings of the 27th ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 723–732.

[Pud] P. Pudlak, *A Note on Using the Detrminant for Proving Lower Bounds on the Size of Linear Circuits*, Electronic Colloquium on Computational Complexity (ECCC), Report 42, 1998.

[RS] R. Raz and A. Shpilka, *Lower bounds for matrix product, in bounded depth circuits with arbitrary gates*, in Proceedings of the 33rd ACM Symposium on Theory of Computing, Crete, Greece, 2001, pp. 409–418.

[Shp] A. Shpilka, *Lower bounds for matrix product*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, Las Vegas, NV, 2001.

[Sil] J. W. Silverstein, *The smallest eigenvalue of a large dimensional Wishart matrix*, Ann. Probab., 13 (1985), pp. 1364–1368.

[Str] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1370–1388

MINIMIZING TOTAL COMPLETION TIME ON PARALLEL MACHINES WITH DEADLINE CONSTRAINTS*

JOSEPH Y.-T. LEUNG† AND MICHAEL PINEDO‡

Abstract. Consider n independent jobs and m identical machines in parallel. Job j has a processing time p_j and a deadline $\bar{d}_j$. It must complete its processing before or at its deadline. All jobs are available for processing at time $t = 0$ and preemptions are allowed. A set of jobs is said to be *feasible* if there exists a schedule that meets all the deadlines; such a schedule is called a feasible schedule. Given a feasible set of jobs, our goal is to find a schedule that minimizes the total completion time $\sum C_j$. In the classical $\alpha \mid \beta \mid \gamma$ scheduling notation this problem is referred to as $P \mid prmt, \bar{d}_j \mid \sum C_j$. Lawler (*Recent Results in the Theory of Machine Scheduling*, in Mathematical Programming: The State of the Art, A. Bachem, M. Grötschel, and B. Korte, eds., Springer, Berlin, 1982, pp. 202–234) raised the question of whether or not the problem is NP-hard. In this paper we present a polynomial-time algorithm for every $m \geq 2$, and we show that the more general problem with m unrelated machines, i.e., $R \mid prmt, \bar{d}_j \mid \sum C_j$, is strongly NP-hard.

Key words. parallel and identical machines, unrelated machines, total completion time, deadline constraints, preemptive scheduling, maximum lateness, polynomial-time algorithm

AMS subject classifications. Primary, 90B35; Secondary, 68C25

DOI. 10.1137/S0097539702406388

1. Introduction.

Consider m identical machines in parallel and n jobs. Job j has a processing time p_j and a deadline $\bar{d}_j$. Job j must complete its processing before or at its deadline $\bar{d}_j$. All jobs are available for processing at $t = 0$ and preemptions are allowed. A set of jobs is said to be *feasible* if there exists a schedule that meets all its deadlines; such a schedule is called a feasible schedule. Given a feasible set of jobs, our objective is to find a schedule that minimizes the total completion time $\sum C_j$. In the 3-field notation $\alpha \mid \beta \mid \gamma$ introduced by Graham et al. (1979), this problem is referred to as $P \mid prmt, \bar{d}_j \mid \sum C_j$.

The special case with m machines and n jobs without deadlines, i.e., $d_j = \infty$ for all j, can be solved via the well-known shortest processing time first (SPT) rule. This rule will always generate a schedule with minimum $\sum C_j$ (see Pinedo (2002)). The SPT rule selects, whenever a machine has been freed, among the remaining jobs the job with the smallest processing time. Since McNaughton (1959) showed that preemptions cannot reduce $\sum C_j$, the SPT rule solves $P \mid prmt \mid \sum C_j$ as well as $P \mid\mid \sum C_j$.

For the special case with a single machine and n jobs with deadlines, Smith (1956) provided an $O(n \log n)$ algorithm that works as follows. Schedule the jobs backwards, starting at time $t = \sum p_j$. From among all the jobs that can finish their processing at time t (i.e., jobs with $\bar{d}_j \geq t$) select the one with the longest processing time. This leaves a set of $n - 1$ jobs to which the same rule can be applied. Preemptions cannot reduce $\sum C_j$ when there is a single machine and all jobs are released at the same time. So Smith's rule solves $1 \mid prmt, \bar{d}_j \mid \sum C_j$ as well as $1 \mid \bar{d}_j \mid \sum C_j$.

*Received by the editors March 30, 2002; accepted for publication (in revised form) April 28, 2003; published electronically August 29, 2003.
http://www.siam.org/journals/sicomp/32-5/40638.html
†Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102 (leung@cis.njit.edu).
‡Stern School of Business, New York University, 40 West Fourth Street, New York, NY 10012 (mpinedo@stern.nyu.edu).

If each job j has, instead of a deadline, a release date r_j (before which it cannot start its processing), then minimizing $\sum C_j$ is NP-hard in the nonpreemptive case but solvable in polynomial time in the preemptive case. Lenstra (1977) showed that $1 \mid r_j \mid \sum C_j$ is NP-hard, and Baker (1974) presented an $O(n \log n)$ algorithm for $1 \mid prmt, r_j \mid \sum C_j$. Baker's rule schedules, at each point in time, the job with the smallest remaining processing time from among all the available jobs.

Lawler (1982) posed the single machine problem with release dates and deadlines, i.e., $1 \mid prmt, r_j, \bar{d}_j \mid \sum C_j$, and raised the question of whether it can be solved in polynomial time. This question has been answered in the negative by Du and Leung (1993), who showed that the problem is NP-hard. Thus, as far as polynomial-time algorithms are concerned, we cannot both have release dates and deadlines in the problem. Hence, we are forced to consider problems either with release dates or deadlines but not both.

Du and Leung (1993) also generalized Baker's rule and Smith's rule to solve a much broader class of problem instances, namely, those that contain no ordered triple of jobs (i, j, k) with

$$r_i, r_k < r_j < \bar{d}_i < \bar{d}_j, \bar{d}_k$$

and $p_j < p_i, p_k$; such a triple is called an obstruction. The generalized Baker's algorithm, the generalized Smith's algorithm, and an algorithm that recognizes problem instances with no obstruction can all be implemented in $O(n^2)$ time. Note that problem instances that contain no obstruction include those for which (1) intervals $[r_j, \bar{d}_j]$, $j = 1, 2, \ldots, n$, are nested; (2) release dates and deadlines are oppositely ordered; (3) processing times and deadlines are similarly ordered; (4) processing times and release dates are similarly ordered; (5) processing times are identical.

Lawler (1982) also raised the question of whether $P \mid prmt, r_j \mid \sum C_j$ and $P \mid prmt, \bar{d}_j \mid \sum C_j$ can be solved in polynomial time. Du, Leung, and Young (1990) showed that the first one of these two problems is NP-hard, even for two identical and parallel machines. In this paper we show that the second of these two problems, i.e., $P \mid prmt, \bar{d}_j \mid \sum C_j$, can be solved in polynomial time for every $m \geq 2$. (As noted before, the single machine case had already been settled by Smith (1956).)

Polynomial-time algorithms that check whether a set of jobs is feasible do exist, even when the jobs have release dates and deadlines. Horn (1974) showed that the problem of determining feasibility can be reduced to a network flow problem. Faster algorithms exist if the jobs have identical release dates or identical deadlines (see Sahni (1979)), or the special case when the intervals $[r_j, \bar{d}_j]$, $j = 1, 2, \ldots, n$, are nested (see Hong and Leung (1989)).

Gonzalez (1978) and McCormick and Pinedo (1995) described a polynomial-time algorithm for a feasible set of jobs that all have a common deadline. As we will see later, the scheduling problem becomes more complex when the jobs have different deadlines.

In the next section we present a polynomial-time algorithm for $P \mid prmt, \bar{d}_j \mid \sum C_j$. In the following section we show that the more general problem with unrelated machines, i.e., $R \mid prmt, \bar{d}_j \mid \sum C_j$, is strongly NP-hard. In the last section we present extensions and our conclusions.

2. The algorithm. In this section we assume that we are given a feasible set of jobs. All schedules are assumed to be feasible unless stated otherwise. Let

$$D_1 < D_2 < \cdots < D_z$$

denote the distinct deadlines of the n jobs, and let $D_0 = 0$. Every schedule S induces a deadline $d_j(S)$ for each job j, where $d_j(S)$ is defined as the smallest D_k with $C_j \leq D_k$ in S. If the context is clear, we drop the S and simply denote the induced deadline by d_j. Note that the induced deadline d_j of job j may be smaller than its original deadline $\bar{d}_j$. However, every feasible schedule has the property that $d_j \leq \bar{d}_j$ for each job j.

We motivate our algorithm by asking two key questions. Suppose a "birdie" were to tell us the induced deadline of each job in an optimal schedule. Having only this information, can we construct an optimal schedule? Second, how do we get the information that the "birdie" has? It is clear that the answers to these two questions immediately yield an optimal algorithm for our problem.

We proceed with answering the first question. The next lemma shows that if a short job has an induced deadline that is no later than that of a long job, then the short job is completed no later (and possibly earlier) than the long job. Lemma 1 can be shown via a standard interchange argument that is omitted here.

LEMMA 1. *Suppose we have two jobs j and k such that $p_j \leq p_k$. If there is a schedule S such that $d_j(S) \leq d_k(S)$, then there is another schedule S' with total completion time $\sum C_j$ not larger than that of S, and job j completes no later (and possibly earlier) than job k in S'.*

Lemma 1 yields a completion sequence in an optimal schedule. That is, jobs with induced deadlines equal to D_1 finish first in ascending order of their processing times, followed by jobs with induced deadlines equal to D_2 in ascending order of their processing times, and so on. Since the jobs complete in this order, they should also be scheduled in the same order. Thus, we consider a list scheduling algorithm with the list L of jobs ordered according to the given completion sequence. Whenever a machine becomes free for assignment, the next job in L will be scheduled in such a way that it completes as early as possible, with the provision that the remaining jobs in L can meet their induced deadlines. We refer to this procedure as the Scheduling Algorithm *SA*. The input into *SA* is the ordering L in which the algorithm will try to complete the n jobs. The output of *SA* is a complete schedule with the starting times, preemptions and completion times of all n jobs. However, the sequence in which the jobs complete their processing in the schedule generated by *SA* may not be identical to the target ordering L. (As we shall see later, the completion sequence will be identical to L if L is an optimal ordering.) The *SA* algorithm consists of eight steps.

Step 1 (initialization). Reindex the jobs so that job 1 is supposed to finish first, job 2 second, and so on. So $L = (1, 2, \ldots, n)$. Let $j = 1$.

Step 2 (computation of machine availabilities). Suppose we have scheduled the first $j-1$ jobs and we consider job j. Let f_i, $i = 1, \ldots, m$, denote the time machine M_i becomes available (is freed) after the first $j-1$ jobs have been completed. Reindex the machines so that

$$f_1 \leq f_2 \leq \cdots \leq f_m.$$

Clearly, if job j were to complete as early as possible, it should be scheduled on machine M_1 beginning at time f_1. However, this may lead to an infeasible schedule. Thus, we need to determine the minimum amount of time that we need to delay job j's starting time in order to have a feasible schedule. The following steps contain a procedure for determining the exact starting time and completion time of job j.

Step 3 (selection of the $m-1$ jobs with least slack). For each job k that follows

job j in L, let LST_k denote its latest possible starting time. Clearly,

$$LST_k = d_k - p_k,$$

i.e., its induced deadline minus its processing time. Job k can meet its induced deadline if and only if it starts at or before LST_k. Let jobs $k_1, k_2, \ldots, k_{m-1}$ be the $m-1$ jobs with the earliest LST_k among all the jobs that follow job j in L, i.e., the $m-1$ jobs with the least slack. (If there are ties, then select the job with the earliest induced deadline.) Reindex these jobs so that they are in ascending order of their latest-starting-times (LSTs); i.e.,

$$LST_{k_1} \leq LST_{k_2} \leq \cdots \leq LST_{k_{m-1}}.$$

Step 4 (computation of deficits and idle times). Assign job k_i to machine M_{i+1}, $i = 1, \ldots, m-1$. If $f_{i+1} \leq LST_{k_i}$ for each $1 \leq i \leq m-1$, then this is feasible. In general, however, it may happen that $f_{i+1} > LST_{k_i}$ for some index i and $f_{i+1} \leq LST_{k_i}$ for some other index i. If $f_{i+1} < LST_{k_i}$, then machine M_{i+1} is said to have an *idle time* and

$$\sigma_{i+1} = LST_{k_i} - f_{i+1}$$

denotes the length of this idle time. If $f_{i+1} > LST_{k_i}$, then machine M_{i+1} is said to have a *deficit* and

$$\delta_{i+1} = f_{i+1} - LST_{k_i}$$

denotes the size of this deficit. To meet the induced deadline of job k_i, we need to schedule an amount δ_{i+1} of job k_i by the time f_{i+1} on machines with an index lower than $i+1$. (Note that machines with index higher than $i+1$ have no idle times prior to f_{i+1} since the machines are indexed in ascending order of their finishing times.) For each deficit machine M_{i+1}, we associate with job k_i a deficit $\hat{\delta}_i$ and a deadline $\hat{d}_i$, where $\hat{\delta}_i = \delta_{i+1}$ and $\hat{d}_i = f_{i+1}$. This signifies that an amount of $\hat{\delta}_i$ of job k_i needs to be processed by the time $\hat{d}_i$ on some other machines.

Step 5 (redistribution of deficits). We now describe a procedure to redistribute the deficits of some jobs to machines with idle times. Let jobs $k_{i_1}, k_{i_2}, \ldots, k_{i_x}$ be all the jobs with deficits and machines $M_{j_1}, M_{j_2}, \ldots, M_{j_y}$ be all the machines with idle times. We assume that $i_1 < i_2 < \cdots < i_x$ and $j_1 < j_2 \cdots < j_y$. Going from job k_{i_x} to job k_{i_1}, we repeat the following steps. Consider job k_{i_a}. Look for the largest indexed machine M_{j_b} that is still lower in index than the machine to which job k_{i_a} is assigned originally; i.e., $j_b < i_a + 1$. We schedule as much as possible of job k_{i_a} on machine M_{j_b}. If $\sigma_{j_b} \geq \hat{\delta}_{i_a}$, then all of the deficits of job k_{i_a} can be scheduled on machine M_{j_b}. In this case we decrement σ_{j_b} by $\hat{\delta}_{i_a}$, delete job k_{i_a} from the list, and repeat the steps with the next job. On the other hand, if $\sigma_{j_b} < \hat{\delta}_{i_a}$, then only a portion of the deficit of job k_{i_a} can be scheduled on machine M_{j_b}, and the rest will have to be redistributed to other machine(s). In this case we decrement both $\hat{\delta}_{i_a}$ and $\hat{d}_{i_a}$ by σ_{j_b}, delete machine M_{j_b} from the list, and repeat the step with job k_{i_a} on the next lower indexed machine (i.e., machine $M_{j_{b-1}}$).

When the above procedure halts, either the list of jobs is exhausted or the list of machines is exhausted. If the list of jobs is exhausted, then we have successfully redistributed the deficits of all the jobs to other machines.

Step 6 (scheduling of job j and remaining deficits on machine 1). If the list of jobs is exhausted, then schedule job j on machine M_1, beginning at time f_1. Otherwise, if

the list of machines is exhausted and there are still some jobs with deficits, schedule the deficits of the remaining jobs on machine M_1, along with job j. Let jobs $k_{i_1}, k_{i_2}, \ldots, k_{i_z}$ be the remaining jobs on the list, with their corresponding deficits and deadlines. Job j has a processing time p_j and an induced deadline d_j. We schedule these $z+1$ "pieces" by the *earliest deadline rule;* i.e., the pieces are scheduled in ascending order of their deadlines. (If the list L corresponds to an optimal completion sequence, there would be no deadline violations in scheduling these $z+1$ pieces. However, if L does not correspond to an optimal completion sequence, then there may be some deadline violations.) If the *earliest deadline* schedule does not generate any deadline violations, then we will attempt to improve the completion time of job j. This is done by moving job j ahead of the pieces that were scheduled before it by the *earliest deadline rule,* provided that it does not generate any deadline violations.

Step 7 (update schedule and other data for next iteration). When job j is settled in the earliest possible position in the above procedure, the pieces that are scheduled before job j will be fixed in their positions, along with job j. We reduce the processing times of the corresponding jobs by the amounts fixed on machine M_1. The pieces that are scheduled after job j on machine M_1 will not be scheduled yet; neither will the jobs scheduled on machines $M_2, M_3, \ldots, M_m$. In other words, the only pieces that are scheduled in this step consist of job j in its entirety as well as all the pieces scheduled before job j; nothing else is scheduled in this step. Since the processing times of some jobs have been reduced, the remaining jobs need be reordered so that jobs with smaller remaining processing times precede jobs with larger remaining processing times in the event that they have identical induced deadlines.

Step 8 (stopping criterion). If $j < n-1$, increase j by 1 and go back to Step 2. If $j = n-1$, schedule the remaining part of the last job in such a way that it is completed as early as possible and STOP (the schedule is complete).

This completes the description of the algorithm. The following example illustrates the algorithm.

Example 1. Consider three machines and 9 jobs.

Jobs	1	2	3	4	5	6	7	8	9
p_j	4	5	7	8	10	12	13	14	15
d_j	∞	∞	∞	10	12	12	25	27	30
LST_j	∞	∞	∞	2	2	0	12	13	15

Apply algorithm *SA* so as to complete the jobs in the order $1, 4, 5, 6, 7, 8, 9, 2, 3$.

The first iteration of *SA* starts with $f_1 = f_2 = f_3 = 0$. Furthermore,

$$LST_{k_1} = LST_6 = 0,$$

$$LST_{k_2} = LST_4 = 2.$$

So job 1 can be scheduled in the interval $[0, 4]$ on machine 1.

The second iteration of *SA* considers job 4. Now $f_1 = f_2 = 0$ and $f_3 = 4$. The two jobs with the earliest LST are jobs 5 and 6, and

$$LST_6 = 0,$$

$$LST_5 = 2.$$

So job 5 is put on machine 3 but can be processed only during the interval $[4, 12]$ since $f_3 = 4$. So there is a deficit of 2. Job 6 is assigned to machine 2 and occupies the interval $[0, 12]$. So $\delta = 2$, and the processing of job 4 on machine 1 has to be postponed 2 time units. So the result of this iteration is that job 5 is processed on machine 1 during interval $[0, 2]$ and job 4 during interval $[2, 10]$. Before going to the next iteration of *SA* the processing time of job 5 has to be adjusted from 10 time units to 8 time units.

The third iteration of *SA* considers job 5. Now $f_1 = 0$, $f_2 = 4$, and $f_3 = 10$. The two jobs with the earliest LST are jobs 6 and 7:

$$LST_6 = 0,$$

$$LST_7 = 12.$$

Job 7 can go on machine 3, and there is an idle time. Job 6 can go on machine 2, and there is a deficit of 4. Job 6 therefore has to be processed on machine 1 during the interval $[0, 4]$ and $\delta = 4$. So job 5 can be processed on machine 1 during the interval $[4, 12]$, and this assignment is feasible. So the result of this iteration is that job 6 is processed during the interval $[0, 4]$ and job 5 during the interval $[4, 12]$. Before moving to the fourth iteration the processing time of job 6 has to be adjusted from 12 to 8.

Continuing in this fashion *SA* can schedule all jobs so that they are completed in the order $1, 4, 5, 6, 7, 8, 9, 2, 3$.

The next theorem is instrumental in proving that *SA* yields an optimal schedule.

THEOREM 1. *Given the order of completions in an optimal schedule, there exists an optimal schedule (i.e., a schedule with minimum* $\sum C_j$*) in which each job is completed as early as possible, provided that all the jobs that follow can meet their induced deadlines.*

Proof. Let $L = (1, 2, \ldots, n)$ denote the completion sequence in an optimal schedule. Let S denote the schedule in which each job is completed as early as possible (provided that all the jobs following it can meet their induced deadlines), and let S' be another schedule that does not complete each job as early as possible. Let j be the smallest index such that job j completes earlier in S than in S'. Let job j complete at time t in S'. Since job j does not finish as early as possible, there must be a noncritical job k (i.e., job k completes before its original deadline), $k > j$, such that job k is processed in the interval $[\bar{t} - \delta, \bar{t})$, $\bar{t} < t$, while job j is not processed in the same interval. Let job k complete at time t' in S'.

Consider the following triple interchange (see Figure 1). Move the piece of job j in the interval $[t - \delta, t)$ to the interval $[\bar{t} - \delta, \bar{t})$ where job k was. Move the piece of job k in the interval $[\bar{t} - \delta, \bar{t})$ to the interval $[t', t' + \delta)$, possibly moving out one job l processed in that interval. Finally, move job l to the interval $[t - \delta, t)$ where job j was. It is clear that the interchange is always feasible if $t' + \delta \leq \bar{d}_k$. After the interchange, job j's completion time goes down by δ, whereas job k's completion time goes up by δ, and job l's completion time cannot go up (and may go down). Thus, the total completion time of the new schedule is less than or equal to that of the old schedule. We can repeat the same argument until job j is completed as early as possible. ∎

THEOREM 2. *Given the completion sequence in an optimal schedule, the algorithm* SA *always constructs an optimal schedule.*

Proof. The scheduling algorithm above schedules each job to complete as early as possible. By Theorem 1, it produces the minimum $\sum C_j$. It remains to be shown that *SA* produces a feasible schedule, given that the completion sequence is feasible.

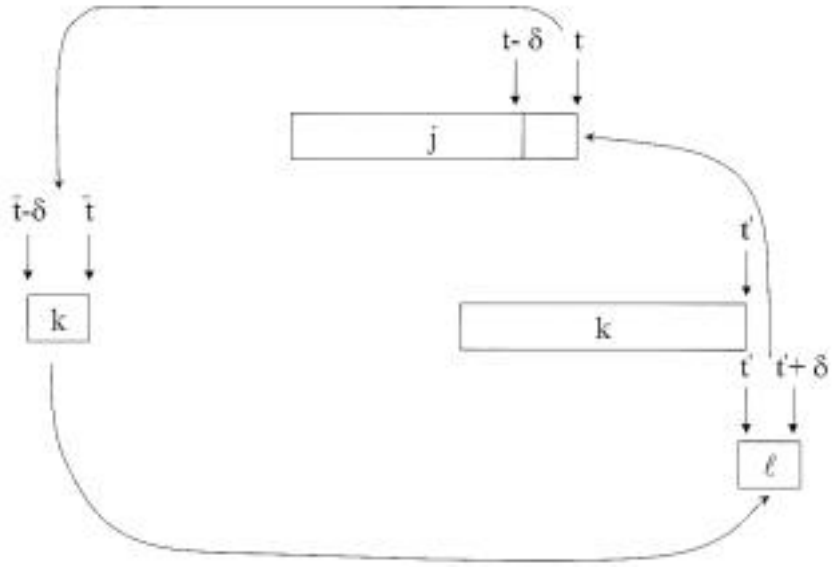


FIG. 1. *Triple interchange.*

SA schedules job j in sequence, $j = 1, 2, \ldots, n$. The proof is by induction on j. The base case $j = 1$ is obvious. Assume that it is true for $j < i$; we will show that it is true for $j = i$.

Recall that SA tries to schedule job i on machine M_1, beginning at time f_1. To check if the remaining jobs can complete by their induced deadlines, the algorithm considers the $m - 1$ jobs with the earliest LSTs, $k_1, k_2, \ldots, k_{m-1}$, where

$$LST_{k_1} \leq LST_{k_2} \leq \cdots \leq LST_{k_{m-1}}.$$

Job k_j is assigned to machine M_{j+1}, $1 \leq j \leq m - 1$, up against its induced deadline. If there is any overlap of job k_j with a job that is already scheduled on machine M_{j+1}, it redistributes the overlapped portion to a machine with a lower index. After the redistribution process, let machine M_l be the lowest indexed machine that has idle time; i.e., machines $M_2, M_3, \ldots, M_{l-1}$ have no idle time; see Figure 2. (If none of these machines have idle time, we define l to be $m + 1$.) Refer to jobs $k_1, k_2, \ldots, k_{l-2}$ as the critical jobs (i.e., the jobs scheduled on machines $M_2, M_3, \ldots, M_{l-1}$) and to jobs $k_{l-1}, k_l, \ldots, k_{m-1}$ as the noncritical jobs. By the induction hypothesis, the $m-1$ jobs as well as job i can be scheduled in a feasible manner.

We want to show that the remaining jobs (other than the $m - 1$ jobs) can also be feasibly scheduled after job i is scheduled. We consider two cases, depending on whether or not $l = m + 1$. If $l = m + 1$, then all $m - 1$ jobs are critical. Since job i must complete before any of the remaining jobs, the remaining jobs can be feasibly scheduled by the induction hypothesis. Thus, we may assume that $l < m + 1$. In this case, job i will start at or before f_l, since the δ amount from the critical jobs (scheduled on machine M_1) cannot extend beyond f_l (or else the critical jobs are not feasibly scheduled). On the other hand, the LST of any of the remaining jobs must

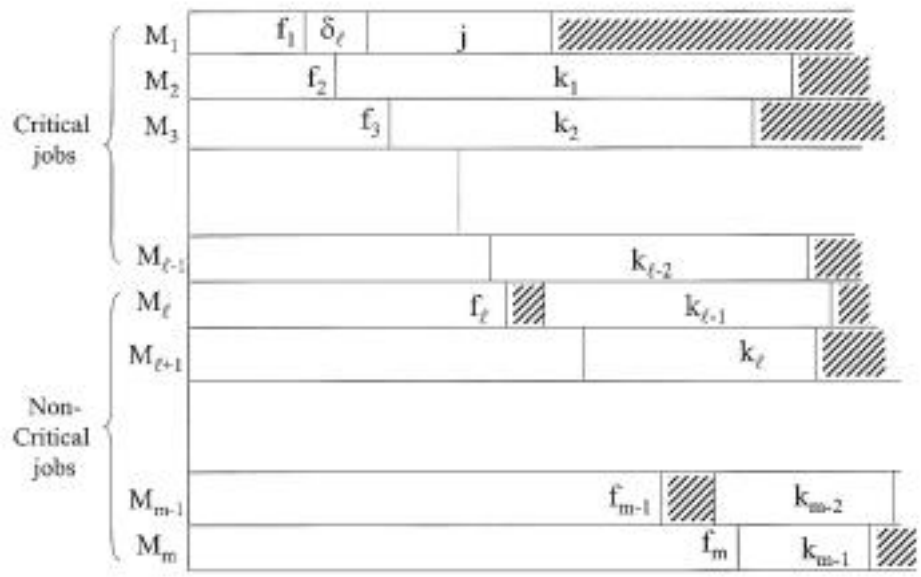


FIG. 2. *Illustrating feasibility.*

be greater than f_l. Thus, all of the remaining jobs as well as the noncritical jobs can be feasibly scheduled if there are enough machine capacities. But this last point is guaranteed since the completion sequence is obtained from an optimal schedule (which is a feasible schedule). □

We now focus on the second question, "How do we get the information the 'birdie' has?" The basic idea is to use the SPT rule, ignoring the deadline constraints. Afterall, if the SPT schedule does not have any deadline violations, then the schedule is already optimal. In general, however, the SPT schedule may have some deadline violations. We need some mechanism to avoid deadline violations while maintaining as much of an SPT-type structure as possible. In what follows we describe the algorithm that generates the optimal ordering, and we refer to this algorithm as the Ordering Algorithm *OA*. (This ordering algorithm uses the *SA* algorithm as a subroutine.) *OA* consists of an initialization step and a main step. The input of this algorithm is the list of jobs L which orders the jobs in increasing order of their processing times and, when there are ties, in increasing order of their original deadlines. The output of the algorithm is a list $\bar{L}$ that specifies the order in which the jobs are completed in an optimal schedule.

Step 1 (initialization). Reindex the jobs in ascending order of their processing times and in ascending order of their original deadlines for identical processing times. Let $L = (1, 2, \ldots, n)$ be the list of jobs in ascending order of their indexes. For each job j, initialize its induced deadline as its original deadline $\bar{d}_j$. (This is a slight abuse of notation since induced deadlines are defined with respect to a schedule. As we shall see later, the induced deadlines are updated to correspond to the induced deadlines in an optimal schedule.) Set $k = 1$ and $\bar{L} = L$.

Step 2 (main). In what follows we reorder the jobs in $\bar{L}$ to form an ordering of the completion sequence of an optimal schedule. We consider each job in turn, starting

with the first job in $\bar{L}$. Suppose we have fixed the position of the first $k-1$ jobs and we are considering the kth job. Let $\bar{L} = (i_1, i_2, \ldots, i_n)$, and let $R(i_{k+1}, i_{k+2}, \ldots, i_n)$ denote the list obtained by reordering the last $n-k$ jobs in $\bar{L}$ in ascending order of their induced deadlines and in ascending order of their processing times for identical induced deadlines. We now construct a schedule S by applying SA to the list

$$\hat{L} = (i_1, i_2, \ldots, i_k) \,||\, R(i_{k+1}, i_{k+2}, \ldots, i_n).$$

The outcome of the application of SA may fall into one of three cases.

Case (i). S is a feasible schedule, and job i_k is completed no later than any one of the jobs in $(i_{k+1}, i_{k+2}, \ldots, i_n)$. In this case job i_k is fixed in position k, and its induced deadline is updated and made equal to its deadline induced by S. $\bar{L}$ will be the same as before (i.e., $\bar{L} = (i_1, i_2, \ldots, i_n)$), and the above process will be repeated with k increased by 1.

Case (ii). S is feasible, but job i_k is completed later than some of the jobs in $(i_{k+1}, i_{k+2}, \ldots, i_n)$. In this case we move all the jobs that are completed earlier than job i_k ahead of job i_k. Again, these jobs (that were moved ahead) will be rearranged in ascending order of their processing times and in ascending order of their induced deadlines for identical processing times. Let L' be the list of these jobs, and let

$$L'' = (i_{k+1}, i_{k+2}, \ldots, i_n) - L'.$$

We set $\bar{L}$ to be

$$\bar{L} = (i_1, i_2, \ldots, i_{k-1}) \,||\, L' \,||\, (i_k) \,||\, L''.$$

The above process will be repeated with the new $\bar{L}$ but the same k. (In other words, k is not increased by 1.)

Case (iii). S is infeasible, and job i_j is the first job that encounters infeasibility when it is scheduled by SA. Recall that SA detects infeasibility when it tries to schedule job i_j on machine 1 along with "pieces" of some other jobs. Let i_x be the job with the earliest induced deadline among all these jobs. In this case we move all the jobs in $(i_{k+1}, i_{k+2}, \ldots, i_n)$ with induced deadlines less than or equal to that of job i_x ahead of job i_k. These jobs (that were moved ahead) will be rearranged in ascending order of their processing times and in ascending order of their induced deadlines for identical processing times. Let L' be the list of these jobs, and let

$$L'' = (i_{k+1}, i_{k+2}, \ldots, i_n) - L'.$$

Set

$$\bar{L} = (i_1, i_2, \ldots, i_{k-1}) \,||\, L' \,||\, (i_k) \,||\, L''.$$

The above process will be repeated with the new $\bar{L}$ but the same k (in other words, k is not increased by 1).

Step 3 (stopping criterion). If $k = n$, then STOP; otherwise go back to Step 2.

When the ordering algorithm stops, $\bar{L}$ specifies the completion sequence of an optimal schedule. The following example illustrates the algorithm.

Example 2. Consider the same three machines and 9 jobs of Example 1.

Jobs	1	2	3	4	5	6	7	8	9
p_j	4	5	7	8	10	12	13	14	15
d_j	∞	∞	∞	10	12	12	25	27	30
LST_j	∞	∞	∞	2	2	0	12	13	15

First iteration of OA ($k = 1$). The input is

$$\bar{L} = 1, 2, 3, 4, 5, 6, 7, 8, 9.$$

The ordering algorithm calls the algorithm *SA* to apply it to the list

$$\hat{L} = 1, 4, 5, 6, 7, 8, 9, 2, 3.$$

This input into *SA* is exactly the instance considered in Example 1, and in Example 1 it is shown how a schedule can be generated according to $\hat{L}$. This completes the first iteration of the ordering algorithm (Case (i)), and job 1 is put in the first position. The induced deadline of job 1 is $d_1(\hat{L}) = 10$, and k is increased by 1.

Second iteration of OA ($k = 2$). The input is the same $\bar{L}$ as in the first iteration since this has not been changed in the first iteration. An attempt has to be made to schedule job 2 (the second shortest job) by applying the *SA* algorithm on

$$\hat{L} = 1, 2, 4, 5, 6, 7, 8, 9, 3.$$

The fitting of job 1 (with $f_1 = f_2 = f_3 = 0$) is similar to the fitting of job 1 in the application of *SA* in the first iteration of the ordering algorithm. *SA* proceeds with the fitting of job 2, given that $f_1 = f_2 = 0$ and $f_3 = 4$. The two jobs with the smallest LST are jobs 4 and 6:

$$LST_6 = 0,$$

$$LST_4 = 2.$$

Machine 3 has a deficit of 2 with job 4, and machine 2 has a deficit of 0 with job 6. So $\delta = 2$. So job 4 has to be processed during the interval $[0, 2]$ on machine 1, and job 2 has to be processed during the interval $[2, 7]$ on machine 1. Before continuing with the third iteration of the *SA* algorithm the processing time of job 4 is modified from 8 to 6.

The third iteration of *SA* tries to fit in job 4 with $f_1 = 0$, $f_2 = 4$, and $f_3 = 7$. The two jobs with the smallest LST are jobs 5 and 6:

$$LST_6 = 0,$$

$$LST_5 = 2.$$

Machine 3 has a deficit of 5 when fitting in job 5, and the deadline of the deficit "piece" of job 5 is 7. Machine 2 has a deficit of 4 when fitting in job 6, and the deadline of the deficit "piece" of job 6 is 4. These two deficits have to be scheduled on machine 1 together with job 4 (which has a processing time of 6 and a deadline of 10). This is infeasible. So the application of *SA* results in Case (iii) of the ordering algorithm and the first job that encounters infeasibility is job 4. Among the three jobs that we tried to schedule on machine 1 (namely jobs 4, 5, and 6), job 4 has the earliest deadline. So this implies that only job 4 is moved before job 2, and the new list is

$$\bar{L} = 1, 4, 2, 3, 5, 6, 7, 8, 9.$$

So we go back to Step 2 of *OA* with $k = 2$.

The remaining iterations of this example have been relegated to the appendix.

The following theorem shows that the algorithm *OA* does generate an optimal completion ordering.

THEOREM 3. *The algorithm* OA *yields an optimal completion ordering* $\bar{L}$.

Proof. Let $L' = (1', 2', \ldots, n')$ be an optimal ordering and S' be an optimal schedule with completion sequence $1', 2', \ldots, n'$. By Theorem 2, we may assume that S' is constructed by *SA* using the list L'. Let $\bar{L} = (1, 2, \ldots, n)$ be the ordering obtained by the above algorithm. Let k be the smallest index such that $k' \neq k$; i.e., $i' = i$ for each $1 \leq i < k$ but $k' \neq k$. We differentiate among three cases, depending upon the processing times of jobs k' and k.

Case I. $p_{k'} < p_k$. Since $p_{k'} < p_k$, job k' appears before job k in the initial ordering of $\bar{L}$, but it appears after job k in the final ordering of $\bar{L}$. This means that job k' was considered in the job ordering process before job k but was overtaken by job k. In the *OA* algorithm, a job will be overtaken by other jobs only if (i) it fails to produce a feasible schedule or (ii) it fails to complete before all of the remaining jobs. In the first case, S' (with job completion sequence identical to L') is not a feasible schedule. In the second case, it can be shown that S' is not optimal.

Case II. $p_{k'} = p_k$. If the original deadline of job k' is greater than or equal to that of job k, i.e., $\bar{d}_{k'} \geq \bar{d}_k$, then we can swap k' with k in L', and the new ordering will produce a feasible schedule with total completion time equal to that of S'. Thus, we may assume that $\bar{d}_{k'} < \bar{d}_k$. In this case job k' appears before job k in the initial ordering of $\bar{L}$, but it appears after job k in the final ordering. Again, this means that job k' was overtaken by job k in the job ordering process. But this is impossible since if it is feasible to complete job k before job k', it must also be feasible to complete job k' before job k. (Recall that $p_{k'} = p_k$ and $\bar{d}_{k'} < \bar{d}_k$.)

Case III. $p_{k'} > p_k$. If $\bar{d}_{k'} \geq \bar{d}_k$, then we can swap k' with k in L', and the new ordering will produce a feasible schedule with total completion time less than that of S'. Thus, we may assume that $\bar{d}_{k'} < \bar{d}_k$. If job k completes by $\bar{d}_{k'}$ in S', then we can swap k' with k in L', and the new ordering will produce a feasible schedule with total completion time less than that of S'. Thus, we may assume that job k completes later than $\bar{d}_{k'}$ (but at or before $\bar{d}_k$) in S'. Let $\bar{d}_{k'} = D_x$ and the induced deadline of job k in S' be D_y. By our assumption, $D_x < D_y$.

Observe that our scheduling algorithm schedules the noncritical jobs nonpreemptively, while the critical jobs (the jobs that were pushed against the deadline and involved in the computation of LST) may be preempted. Consider the jobs that follow job k' in L' up until job k. We assert that there is a noncritical job j with its original deadline $\bar{d}_j > D_{y-1}$ and job j starts before D_{y-1} in S'. This is because $\bar{L}$ (with job k in the kth position) indicates that it is possible to complete job k along with all the jobs whose original deadlines are less than or equal to D_{y-1} by D_{y-1}. Since job k does not complete by D_{y-1} in S', there must be another job in its place. We consider two cases, depending upon the processing times of jobs j and k.

If $p_j < p_k$, then job j was considered in the job ordering process before job k, but it was overtaken by job k. But this is impossible since S' indicates that it is feasible to complete job j before job k.

If $p_j \geq p_k$, then we can swap j with k in L', and the new ordering will produce a feasible schedule with total completion time less than or equal to that of S'. We can repeat the above argument until job k completes by $\bar{d}_{k'}$, at which time we can swap job k' with job k. □

Theorems 2 and 3 yield our main result.

THEOREM 4. *$P \mid prmt, \bar{d}_j \mid \sum C_j$ can be solved in polynomial time for each $m \geq 2$.*

The running time of our algorithm is $O(mn^3 \log mn)$. Step 1 (initialization) of algorithm OA takes $O(n \log n)$ time. Step 2 (main) of OA takes at most $O(n^2)$ iterations, since in the worst case the algorithm may completely reverse the initial ordering. Each iteration involves rearranging the last $n-k$ jobs in ascending order of their induced deadlines, calling SA with the list $\hat{L}$, and modifying $\bar{L}$ if the outcome of the application of SA falls into Cases (ii) or (iii). In what follows we show that each of these steps can be implemented in $O(mn \log mn)$ time.

The most efficient way to rearrange the last $n-k$ jobs in ascending order of their induced deadlines is to maintain a balanced tree of the remaining jobs (i.e., the last $n-k$ jobs) in ascending order of their induced deadlines (and in case of ties in ascending order of their processing times). This tree can initially be built in $O(n \log n)$ time. When we need the required ordered list, we simply walk over the tree in linear time. As k increases, we need to delete some jobs from the balanced tree. But this can all be done in the required time bound $O(mn \log mn)$.

The algorithm SA can be implemented in $O(mn \log mn)$ time. There are n jobs to schedule, and each job can be scheduled in $O(m \log m + m \log n)$ time. The machines can be sorted in ascending order of their finishing times in $O(m \log m)$ time. The $m-1$ jobs with the earliest LSTs can be determined in $O(m \log n)$ time if we maintain a balanced tree of jobs in ascending order of their LST. The redistribution of deficits of some jobs to other machines takes $O(m)$ time, and the scheduling of the deficits on machine 1 takes the same amount of time.

Modifying $\bar{L}$ can also be implemented to run in the required time bound. Identifying the jobs in L' can be done in linear time. Sorting them in ascending order of their processing times (and in case of ties in ascending order of their induced deadlines) takes $O(n \log n)$ time. Deleting the jobs in L' from the list $(i_{k+1}, i_{k+2}, \ldots, i_n)$ takes linear time if $\bar{L}$ is maintained as a doubly linked list. Thus, the new $\bar{L}$ can be obtained in $O(n \log n)$ time.

3. Complexity results. In this section we show that generalizing $P \mid prmt, \bar{d}_j \mid \sum C_j$ to a parallel machine environment with machines that are unrelated results in a problem that is strongly NP-hard. (Recall that in a parallel machine environment that is unrelated the processing time of job j on machine i is p_{ij}; i.e., the processing time of job j depends on the machine on which it is processed.)

Our proof that shows the NP-hardness of $R \mid prmt, \bar{d}_j \mid \sum C_j$ also shows the NP-hardness of the nonpreemptive $R \mid \bar{d}_j \mid \sum C_j$ problem. It shows that, even when all the jobs have a single common deadline, the problem is NP-hard. In what follows we first present the proof for the nonpreemptive environment and then explain why it also applies to the preemptive environment.

We shall reduce the strongly NP-complete 3-DIMENSIONAL MATCHING (3DM) problem to the decision version of $R \mid \bar{d}_j \mid \sum C_j$. The 3DM problem can be stated as follows (see Garey and Johnson (1979)): Given three disjoint sets $A = \{a_1, a_2, \ldots, a_q\}$, $B = \{b_1, b_2, \ldots, b_q\}$, and $C = \{c_1, c_2, \ldots, c_q\}$ and a set $M = \{m_1, m_2, \ldots, m_l\}$ with each m_k, $1 \leq k \leq l$, being a triple with the first component drawn from A, the second from B, and the third from C, is there a subset M' of M such that every element in A, B, and C occurs in exactly one of the triples in M'?

Given an instance of the 3DM problem, we construct an instance of the decision version of $R \mid \bar{d}_j \mid \sum C_j$ as follows. Let there be l machines, where machine i

corresponds to the triple m_i. For each a_j in A, construct a job A_j with processing times $p_{ij} = 1$ if a_j is in m_i; otherwise, $p_{ij} = 2$. Similarly, for each b_j (resp., c_j) in B (resp., C), construct a job B_j (resp., C_j) with processing times $p_{ij} = 1$ if b_j (resp., c_j) is in m_i; otherwise, $p_{ij} = 2$. In addition, create $l - q$ dummy jobs $D_1, D_2, \ldots, D_{l-q}$. The processing times of each of the dummy jobs are 6, independent of the machine on which they are processed. The deadlines of all the jobs (A, B, C, and D jobs) are 6. Finally, let $\omega = 6l$ be the threshold of $\sum C_j$.

We first show that there is a feasible, nonpreemptive schedule for this set of jobs, regardless of whether the instance of 3DM has a matching. Schedule the A, B, and C jobs on q machines, three jobs per machine. The completion time of every job is no larger than 6. Schedule the dummy jobs on the remaining machines, one job per machine. The completion time of each dummy job is 6. Thus, every job meets its deadline.

The next lemma shows that the given instance of 3DM has a solution if and only if the constructed instance of $R \mid \bar{d}_j \mid \sum C_j$ has a solution.

LEMMA 2. *There is a matching if and only if there is a schedule with* $\sum C_j \leq \omega$.

Proof. Suppose there is a matching. Let $m_{i1}, m_{i2}, \ldots, m_{iq}$ be the triples that constitute the matching. We schedule the jobs as follows. Let m_{ik}, $1 \leq k \leq q$, contain a_x, b_y, and c_z. Assign jobs A_x, B_y, and C_z to machine ik. The completion times of these three jobs are 1, 2, and 3, respectively. Assign the dummy jobs D_j to an unassigned machine, one job per machine. The completion time of each dummy job is 6. This schedule has a total completion time

$$\sum C_j = 6(l - q) + q + 2q + 3q = \omega.$$

Conversely, suppose there is a feasible schedule S with $\sum C_j \leq \omega$. We will show that there is a matching. Since the dummy jobs have deadlines 6 and their processing times are 6 on any machine, they must be assigned one job per machine. The total completion time of all the dummy jobs is $6(l - q)$. A machine with a dummy job assigned can no longer be used for assignment, since all jobs have deadlines less than or equal to 6. Let machines $i1, i2, \ldots, iq$ be the machines still available. The A, B, and C jobs must be assigned to these machines. Since S has $\sum C_j \leq \omega$, the total completion time of all of the A, B, and C jobs is less than or equal to $6q$. But this can be attained only if all of these jobs have processing time 1 unit on the machine to which it is assigned, which means that there is a matching.

The same reduction works also for $R \mid prmt, \bar{d}_j \mid \sum C_j$ since preemption cannot reduce the $\sum C_j$ in the constructed instance. □

The following theorem follows immediately from the previous lemma.

THEOREM 5. *$R \mid prmt, \bar{d}_j \mid \sum C_j$ and $R \mid \bar{d}_j \mid \sum C_j$ are both strongly NP-hard.*

4. Extensions and conclusions. In this paper we presented a polynomial-time algorithm for $P \mid prmt, \bar{d}_j \mid \sum C_j$, and we showed that the more general $R \mid prmt, \bar{d}_j \mid \sum C_j$ problem is strongly NP-hard. Note that our algorithm also solves the single machine case (which had been solved by Smith (1956)); i.e., $1 \mid \bar{d}_j \mid \sum C_j$. Recently, we learned that Sitters (2001) had independently shown that the $R \mid prmt \mid \sum C_j$ problem is strongly NP-hard. While Sitters's result implies the strong NP-hardness of $R \mid prmt, \bar{d}_j \mid \sum C_j$, its proof is significantly more complex than ours.

The polynomial-time algorithm presented in this paper can be used to solve other scheduling problems as well. Suppose that, instead of a deadline, each job j has a due date d_j, and the objective is to minimize the maximum lateness, where the lateness

of a job is defined to be the difference between its completion time and its due date. (In the 3-field notation, this problem is denoted by $P \mid prmt \mid L_{\max}$.) This problem can be solved as follows. Parametrize on the maximum lateness. Assume $L_{\max} = z$, and create for all jobs the deadlines $d_j + z$. We then check if there is a feasible schedule with this set of deadlines. The optimal value for the maximum lateness can be obtained by conducting a binary search of z in a range between a lower and an upper bound. Once the minimal value of z has been obtained, say z^*, we can use the algorithm described in this paper to find a schedule that minimizes $\sum C_j$. In this way we can solve the problem of minimizing $\sum C_j$ subject to the constraint that $L_{\max}$ is minimum. Of course, the algorithm will also work for any $L_{\max}$ greater than or equal to z^*.

Appendix. This appendix contains the remaining iterations of Example 2.

Third iteration of OA ($k = 2$). The algorithm *SA* has to be applied now on

$$\hat{L} = 1, 4, 5, 6, 7, 8, 9, 2, 3.$$

Jobs 1 and 4 are inserted in the same way as they are inserted in Example 1. After having put jobs 1 and 4 in, the partial schedule has job 1 processed during the interval $[0, 4]$, job 5 during the interval $[0, 2]$, and job 4 during the interval $[2, 10]$. So $f_1 = 0$, $f_2 = 4$, and $f_3 = 10$. In the third iteration of this application of the scheduling algorithm job 5 is considered. The two jobs with the smallest LST are jobs 6 and 7 with

$$LST_6 = 0,$$

$$LST_7 = 12.$$

Job 7 can be assigned to machine 3 with an idle time of two time units. Job 6 can be assigned to machine 2 with a deficit of four time units. So $\delta = 4$, and it is possible to put job 5 (with processing time $10 - 2 = 8$ and deadline 12) on machine 1 when $\delta = 4$. So after this iteration of the scheduling algorithm job 6 is processed during the interval $[0, 4]$ (and has a remaining processing time of 8), and job 5 is processed during the interval $[4, 12]$. The machines are available at $f_1 = 4$, $f_2 = 10$, and $f_3 = 12$.

In the next iteration the scheduling algorithm tries to put in job 6 (with a processing time of 8). The two jobs with the smallest LST are jobs 7 and 8:

$$LST_7 = 12,$$

$$LST_8 = 13.$$

Putting in job 8 on machine 3 has an idle time of 1, and putting in job 7 on machine 2 has an idle time of 2. So $\delta = 0$. Putting in job 6 on machine 1 is feasible. So job 6 is processed during the interval $[4, 12]$. The machines are available at $f_1 = 10$, $f_2 = 12$, and $f_3 = 12$.

In the next iteration the scheduling algorithm tries to put in job 7 (with a processing time of 13). The two jobs with the smallest LST are jobs 8 and 9:

$$LST_8 = 13,$$

$$LST_9 = 15.$$

Both jobs 8 and 9 can be scheduled on machines 2 and 3 with idle times. Job 7 can be scheduled on machine 1. The scheduling algorithm can complete the schedule in a feasible manner.

The result of the application of the scheduling algorithm is that job 4 is completed at 10, and job 4 is completed before any one of the jobs following it (i.e., Case (i) of algorithm *OA*). The counter k is increased by 1, and the list $\bar{L}$ remains the same; i.e.,

$$\bar{L} = 1, 4, 2, 3, 5, 6, 7, 8, 9.$$

Fourth iteration of OA ($k = 3$). The scheduling algorithm has to be applied to the list

$$\hat{L} = 1, 4, 2, 5, 6, 7, 8, 9, 3.$$

Job 1 can be scheduled as usual. Job 4 has to be considered with $f_1 = f_2 = 0$ and $f_3 = 4$. Jobs 5 and 6 have the smallest LST:

$$LST_6 = 0,$$

$$LST_5 = 2.$$

Two time units of job 5 have to be processed on machine 1 during interval $[0, 2]$. Job 4 can be processed on machine 1 during interval $[2, 10]$. The remaining processing time of job 5 is 8 and $f_1 = 0$, $f_2 = 4$, and $f_3 = 10$. Job 2 is considered next. Jobs 5 and 6 again have the smallest LST:

$$LST_6 = 0,$$

$$LST_5 = 4.$$

Putting jobs 5 and 6 on machines 2 and 3 result in a total deficit of $6 + 4 = 10$. However, these deficits can be scheduled feasibly on machine 1. Job 6 can be processed on machine 1 during the interval $[0, 4]$ and job 5 during the interval $[4, 10]$, implying that job 2 can start at 10 and is completed at 15. The result of this application of the scheduling algorithm is that the schedule is feasible but that job i_k is completed after some of the jobs in set $(i_{k+1}, \ldots, i_n)$, namely jobs 5 and 6. So jobs 5 and 6 have to be moved before job 2 (Case (ii) of *OA*). So

$$\bar{L} = 1, 4, 5, 6, 2, 3, 7, 8, 9.$$

Fifth iteration of OA ($k = 3$). The scheduling algorithm has to be applied to the list

$$\hat{L} = 1, 4, 5, 6, 7, 8, 9, 2, 3.$$

But this is the same list as in the first iteration of the ordering algorithm. So the schedule is feasible, and job 5 is completed no later than any of the jobs following it (Case (i) of *OA*). So job 5 is fixed in the third position, its induced deadline remains the same, $\bar{L}$ remains the same, and k is increased by 1.

Sixth iteration of OA ($k = 4$). The scheduling algorithm has to be applied to the list

$$\hat{L} = 1, 4, 5, 6, 7, 8, 9, 2, 3.$$

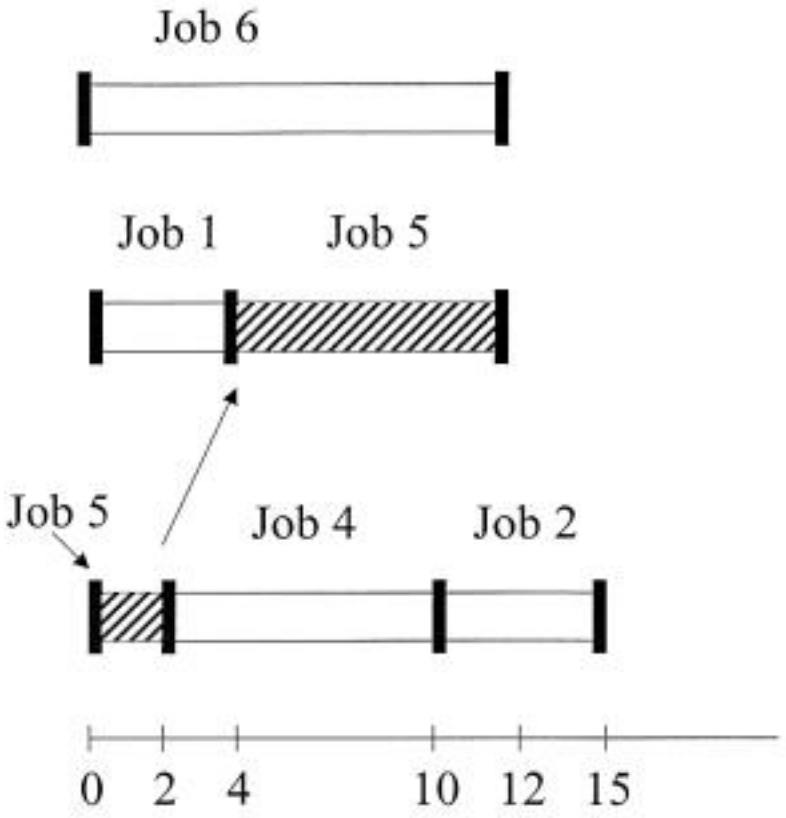


FIG. 3. *Partial schedule.*

This $\hat{L}$ is the same as in the previous iteration, and job 6 is again completed no later than any of the jobs following it and is fixed in the fourth position. Its induced deadline remains the same. $\bar{L}$ remains the same and k is increased by 1.

Seventh iteration of OA ($k = 5$). The scheduling algorithm has to be applied to the list

$$\hat{L} = 1, 4, 5, 6, 2, 7, 8, 9, 3.$$

The scheduling algorithm generates a feasible schedule with job 2 completed before any one of the jobs following it (see Figure 3). The new induced deadline of job 2 is $d_2(\hat{L}) = 25$. $\bar{L}$ remains the same and k is increased by 1.

Eighth iteration of OA ($k = 6$). The scheduling algorithm has to be applied to the list

$$\hat{L} = 1, 4, 5, 6, 2, 3, 7, 8, 9.$$

The ready times of the three machines, when trying to schedule job 3, are $f_1 = 12$, $f_2 = 12$, and $f_3 = 15$. When trying to put in job 3 the jobs with the smallest LST

are jobs 7 and 8:

$$LST_7 = 12,$$

$$LST_8 = 13.$$

Putting jobs 7 and 8 on machines 2 and 3 result in a deficit of 2 because of job 8. So job 8 has to be scheduled on machine 1 during the interval $[12, 14]$, and job 3 can then be processed during the interval $[14, 21]$. So before going to the next step of the scheduling algorithm the processing time of job 8 is updated and becomes 12, and its LST is updated and becomes 15. The next step of the scheduling algorithm tries to put in job 7, while $f_1 = 12$, $f_2 = 15$, and $f_3 = 21$. The two jobs with the smallest LST are jobs 8 and 9:

$$LST_8 = 15,$$

$$LST_9 = 15.$$

It turns out that job 7 cannot be feasibly scheduled. So the eighth iteration of the ordering algorithm ends up in Case (iii). Between the two jobs that we tried to schedule on machine 1 (namely jobs 7 and 9), job 7 has the earliest induced deadline. So job 7 has to be put before job $i_6 = 3$, i.e.,

$$\bar{L} = 1, 4, 5, 6, 2, 7, 3, 8, 9.$$

The k remains the same.

Ninth iteration of OA ($k = 6$). The scheduling algorithm has to be applied to the list

$$\hat{L} = 1, 4, 5, 6, 2, 7, 8, 9, 3.$$

Applying the scheduling algorithm on this order results in a feasible schedule; i.e., it leads to Case (i) in algorithm *OA*. This implies that $\bar{L}$ remains the same. The induced deadline of job 7 has to be updated, but the update does not result in a different deadline. The counter k is increased by 1.

Tenth iteration of OA ($k = 7$). The scheduling algorithm has to be applied to the list

$$\hat{L} = 1, 4, 5, 6, 2, 7, 3, 8, 9.$$

The scheduling algorithm has to try to put in job 3. The times the machines become available after completing the processing of the first six jobs are $f_1 = 12$, $f_2 = 15$, and $f_3 = 25$. There are two jobs listed after job 3 and

$$LST_8 = 13,$$

$$LST_9 = 15.$$

Both machines have a deficit. Machine 2 has a deficit of 2, and machine 3 has a deficit of 10. However, both these deficits can be scheduled feasibly on machine 1, and job 3

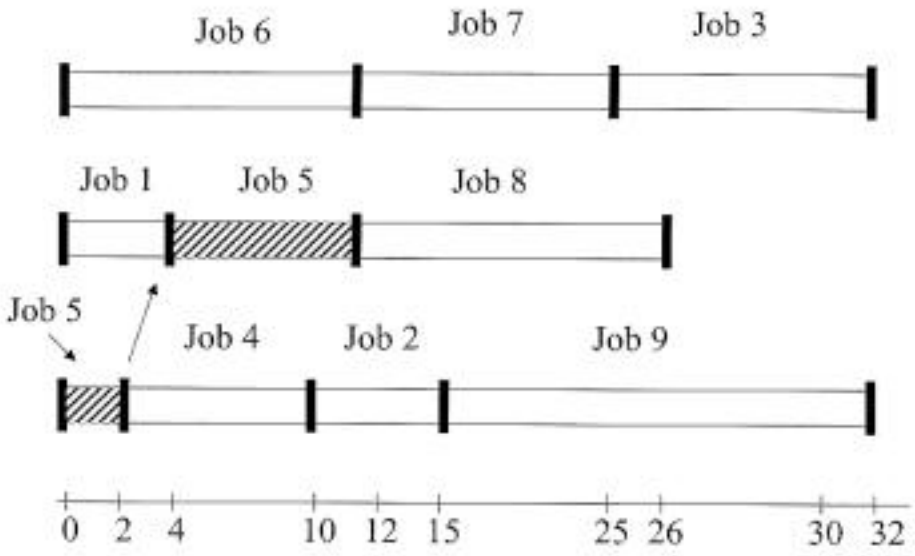


FIG. 4. *Optimal schedule.*

can be scheduled (feasibly) thereafter. This iteration of *OA* ends up in Case (ii). We have to change $\bar{L}$, and it now becomes

$$\bar{L} = 1, 4, 5, 6, 2, 7, 8, 9, 3.$$

The counter k remains the same.

The algorithm needs another two iterations: one for $k = 7$ and one for $k = 8$. In the eleventh iteration ($k = 7$) job 8 is feasibly scheduled, and in the twelfth iteration ($k = 8$) job 9 is feasibly scheduled. The optimal schedule is then obtained by putting in job 3 on the machine that is first available (see Figure 4).

REFERENCES

K. R. BAKER (1974), *Introduction to Sequencing and Scheduling*, Wiley, New York.

J. DU AND J. Y-T. LEUNG (1993), *Minimizing Mean Flow Time with Release Time and Deadline Constraints*, J. Algorithms, 14, pp. 45–68.

J. DU, J. Y-T. LEUNG, AND G. H. YOUNG (1990), *Minimizing mean flow time with release time constraint*, Theoret. Comput. Sci., 75, pp. 347–355.

M. R. GAREY AND D. S. JOHSON (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York.

T. GONZALEZ (1978), *Minimizing the Mean and Maximum Finishing Time on Identical Processors*, Technical report CS-78-15, Department of Computer Science, Pennsylvania State University, University Park, PA.

R. L. GRAHAM, E. L. LAWLER, J. K. LENSTRA, AND A. H. G. RINNOOY KAN (1979), *Optimization and approximation in deterministic sequencing and scheduling: A survey*, Ann. Discrete Math., 5, pp. 287–326.

K. S. HONG AND J. Y-T. LEUNG (1989), *Preemptive scheduling with release times and deadlines*, J. Real-Time Systems, 1, pp. 265–281.

W. A. HORN (1974), *Some simple scheduling algorithms*, Naval Res. Logist. Quart., 21, pp. 177–185.

E. L. LAWLER (1982), *Recent Results in the Theory of Machine Scheduling*, in Mathematical Programming: The State of the Art, A. Bachem, M. Grötschel, and B. Korte, eds., Springer, Berlin, pp. 202–234.

J. K. Lenstra (1977), *Sequencing by Enumerative Methods*, Mathematical Centre Tracts 69, Mathematisch Centrum, Amsterdam, Netherland.

S. T. McCormick and M. Pinedo (1995), *Scheduling n independent on m uniform machines with both flow time and makespan objectives: A parametric analysis*, ORSA J. Comput., 7, pp. 63–77.

R. McNaughton (1959), *Scheduling with deadlines and loss functions*, Management Sci. 6, pp. 1–12.

M. Pinedo (2002), *Scheduling: Theory, Algorithms and Systems*, Prentice-Hall, Englewood Cliffs, NJ.

S. Sahni (1979), *Preemptive scheduling with due dates*, Oper. Res., 27, pp. 925–934.

R. A. Sitters (2001), *Two NP-hardness results for preemptive minsum scheduling of unrelated parallel machines*, in Proceedings of the 8th International IPCO Conference, Lecture Notes in Comput. Sci. 2081, Springer, Berlin, pp. 396–405.

W. E. Smith (1956), *Various optimizers for single-stage production*, Naval Res. Logist. Quart. 3, pp. 59–66.

SIAM J. COMPUT.
Vol. 32, No. 5, pp. 1389–1394

KOLMOGOROV COMPLEXITY AND DETERMINISTIC CONTEXT-FREE LANGUAGES*

OLIVER GLIER†

Abstract. We deal with a criterion for deterministic context-free languages that was originally formulated by Li and Vitányi [*SIAM J. Comput.*, 24 (1995), pp. 398–410]. Their result—called the KC-DCF lemma—relates Kolmogorov complexity to pushdown automata and works on a superset of examples compared to traditional iteration and pumping lemmas. Sadly, their KC-DCF lemma has a flaw. In this paper, we give a counterexample to the original KC-DCF lemma and also provide a corrected version.

Key words. formal language theory, Kolmogorov complexity, deterministic context-free languages

AMS subject classifications. 68Q30, 68Q45

DOI. 10.1137/S0097539702417754

First we settle some notation and give a quick introduction to Kolmogorov complexity as far as it is needed for our result (for more details and applications of Kolmogorov complexity, see [LiVi97]). Let Σ be an alphabet with at least two distinct symbols 0 and 1. For a word $x = x_1 \cdots x_n \in \Sigma^*$ we define

$$x^{\$} := 1^{|\mathtt{bin}(n)|}0\mathtt{bin}(n)x$$

as the *self-delimiting code* for x, where $\mathtt{bin}(n)$ is the binary representation of the number n. In what follows, let U be a universal machine which expects some input of the form $p^{\$}q$, $p, q \in \Sigma^*$ and then simulates the program p on input q. Now we define the following for $x, q \in \Sigma^*$:

$$C(x|q) := \min\{|p| : U(p^{\$}q) = x, p \in \Sigma^*\}$$

is the *conditional Kolmogorov complexity* of x with the additional information q, and

$$C(x) := \min\{|p| : U(p^{\$}) = x, p \in \Sigma^*\}$$

is the *Kolmogorov complexity* of x (without additional information).

The following immediate consequences are stated without proof.

PROPOSITION 1.

(i) $C(x|q) \leq C(x) \leq |x| + O(1)$.

(ii) *For any totally recursive function* $f : \Sigma^* \to \Sigma^*$, *we have*

$$C(f(x)|q) \leq C(x|q) + O(1).$$

(iii) $C(x) \leq C(q) + C(x|q) + O(\min(\log|x|, \log|q|))$. □

For any natural number $n = 0, 1, 2, \ldots$, we denote by $\underline{n}$ the nth word of Σ^* in lexicographic order. Further, we set $\lg(n) := |\underline{n}|$. Informally, the following proposition states that any sufficiently large number m can be enclosed by the length of a number r and by r's much smaller Kolmogorov complexity.

*Received by the editors November 13, 2002; accepted for publication (in revised form) May 27, 2003; published electronically August 29, 2003.

http://www.siam.org/journals/sicomp/32-5/41775.html

†Research Group for Automata Theory and Formal Languages, Department of Computer Science, Technische Universität Darmstadt, Darmstadt, Germany (glier@iti.informatik.tu-darmstadt.de).

PROPOSITION 2. *For sufficiently large m, there always exists an $r \in \mathbb{N}$ with $\lg(r) > m$ and $C(\underline{r}) < \lg(\lg(m))$.*

Proof. We choose $t = \lg(\lg(m))$ and $r = f(f(f(t)))$, where $f : \mathbb{N} \to \mathbb{N}$ maps a number n to the (unique) number n' such that $\underline{n'}$ is the word 1^{n+1}. By applying Proposition 1(ii), we get

$$C(\underline{r}) \leq C(\underline{t}) + O(1) \leq \lg(t) + O(1) < t = \lg(\lg(m))$$

for sufficiently large m. On the other hand, if m is large enough, we have $f(\lg(m)) > m$ and $\lg(f(m)) > m$. It follows that $\lg(r) > m$. □

We say that a word $x \in \Sigma^*$ is *compressible* iff $C(x) < |x|$; otherwise, it is *incompressible*. Similarly, a number $n \in \mathbb{N}$ is compressible if the nth word in lexicographic enumeration $\underline{n}$ is compressible. For $n \in \mathbb{N}$ there are only $2^n - 1$ many words of length shorter than n. Therefore we have the following result.

THEOREM 3 (incompressibility theorem). *For each $n \in \mathbb{N}$ there exists at least one incompressible word of length n. Hence, there exist infinitely many incompressible words.*

1. The KC-DCF lemma. For a language $L \subseteq \Sigma^*$ and a word $x \in \Sigma^*$, we denote by $x^{-1}L := \{v \in \Sigma : vx \in L\}$ the set of words v that extend x to a word in L. A context-free language $L \subseteq \Sigma^*$ is *deterministic* context-free if there exists a deterministic pushdown automaton (dpda) that recognizes L. Sometimes an iteration or pumping lemma can be applied in order to prove that a given context-free language is not deterministic context-free. However, those lemmas are difficult to handle. In [LiVi95] the authors use the incompressibility argument in order to give a more intuitive, necessary criterion for deterministic context-free languages. Since the formulation of their KC-DCF lemma is considerably complicated, they also derive a weaker corollary which is easier to apply. Sadly, the formulation and the proof of the KC-DCF lemma contain some mistakes that also affect the corollary and hence invalidate them both. Before we give an alternative formulation for both the lemma and its corollary, we will discuss the original formulation of the corollary. By giving a counterexample, we disprove the original corollary and—consequently—the original KC-DCF lemma.

Let $x, y \in \Sigma^*$ and c be any constant, and let ω be a recursive sequence over Σ^*. The idea is to repeat y in the input of a dpda that recognizes L, while limiting the amount of information that the dpda keeps on its stack. Eventually, too much information is lost in order to properly unwind the stack. Li and Vitányi's corollary of the KC-DCF lemma states the following: There exists a constant c' such that, for $u, v, w \in \Sigma^*$, where u is a suffix of the left-infinite word $\cdots yyx$ and v is a prefix of ω, the following three conditions imply $C(w) \leq c'$:

(1) $C(v|p_{uv'}) \leq c$ for all prefixes v' of v and programs $p_{uv'}$ that list $(uv')^{-1}L$ in lexicographic order;
(2) $C(w|p_{uv}) \leq c$ for all programs p_{uv} that list $(uv)^{-1}L$ in lexicographic order;
(3) $C(v) \geq 2 \cdot \lg(\lg(|u|))$.

Remark. In [LiVi95], condition (1) is restricted to only the prefix $v' = \varepsilon$ as the empty word, omitting all other prefixes of v. However, then the corollary is not a direct consequence of the original KC-DCF lemma anymore, since it will not suffice in order to reconstruct all required configurations of the dpda in the corresponding condition of the lemma. However, this is not the essential mistake in [LiVi95] (which we point out later in our proof of the KC-DCF lemma). Condition (1) above yields a statement which is logically implied by the KC-DCF lemma and its corollary. The

following counterexample disproves this statement and therefore the original KC-DCF lemma and its corollary.

We show now that there exist deterministic context-free languages L and $u, v, w \in \Sigma^*$ satisfying (1), (2), (3), but $C(w) \geq c'$ for any constant c'.

EXAMPLE 1. *Set $u = 0^{2n}$, $v = 1^n$, $w = 0^n$, and $L = \{0^n 1^m 0^k : n = m + k\}$. Now, for all prefixes v' of v, v is the first half of the lexicographically first word in $(uv')^{-1}L$ which is all 1, while w is the lexicographically first word in $(uv)^{-1}L$ which is all 0. Thus (1) and (2) are satisfied. Now let $c' > 0$ be any constant, and choose an incompressible n with $C(w) > c'$. If n is sufficiently large, we get $C(v) > \log |u|$ and therefore (3). However, L is obviously deterministic context-free.*

From now on we assume that $L \subseteq \Sigma^*$ is an arbitrary nontrivial deterministic context-free language, and that A is a dpda that recognizes L. Without loss of generality (see [Ha78]), we assume that A always reads its entire input. Also, whenever we don't explicitly mention the initial configuration, we assume that the dpda A starts on the input with its initial state and with an empty stack.

The following proposition is already implicitly used in [LiVi95].

PROPOSITION 4. *Given any $x, y \in \Sigma^*$, there exists a constant $c_0 \in \mathbb{N}$ such that the following proposition holds. Let r be some arbitrary positive integer, and let $u \in \Sigma^*$ be any suffix of the (left-infinite) word $\cdots yyx$ such that, after processing u, the dpda's stack size is $c_0 + k$ for some $k \geq 0$. Then there exists another suffix u' of $\cdots yyx$ with length $|u'| = r + s$ for some $s \leq c_0$ such that, after the inputs u and u' are processed, the state of A is the same for each, and the segment of the k topmost stack symbols is also the same in each case.*

The proof is rather technical but straightforward and is therefore omitted here. Also, the proof is already sketched in [LiVi95].

We will now proceed directly with a corrected version of the KC-DCF lemma. For $v, v_1, v_2 \in \Sigma^*$, we call v_1, v_2 a *splitting* of v iff $v_1 v_2 = v$.

PROPOSITION 5 (KC-DCF lemma, corrected). *Let $x, y \in \Sigma^*$ and c be a constant. Then there exists a constant c' with the following property: Let $u, v, w \in \Sigma^*$, where u is a suffix of the left-infinite word $\cdots yyx$. Then the following three conditions imply $C(w) \leq c'$:*

(i) *$C(v''|k^{\$}q) \leq c$ for all splittings $v'v'' = v$. Here, k, q is any pair of stack and state such that, starting with k, q on input v'', the behavior of A is essentially the same as when starting from the configuration after processing uv' and then continuing with v''.*

(ii) *$C(w|k^{\$}q) \leq c$, where k and q are stack and state after processing uv.*

(iii) *$C(v) \geq 2 \cdot \lg(\lg(|u|))$.*

In (i), "essentially the same behavior" of the dpda A on input v'' means that, by observing only the dpda's state, its current position in v'', and its operations on the topmost stack symbol—while ignoring the rest of the stack(!)—the observed sequences are the same each time, whether the dpda started with k, q as stack and state or with the stack and state reached after processing uv'.

Proof of the KC-DCF lemma. Let c_0 be the constant from Proposition 4. We distinguish two cases for the pairs (u, v). In the proof in [LiVi95], infinitely many pairs of (u, v) have been missed due to an improper formulation of the defining conditions for the two different cases. This is also the reason why our conditions in the KC-DCF lemma are different.

Case 1. Assume that for all but finitely many pairs (u, v) which satisfy (i) and (iii) the size of the stack shrinks below c_0 during processing of the v-part of input uv.

Then there exists a constant m such that, for each pair (u, v), the stack has size less than m in at least one step i during processing of v. Let v' be the the part of v which has already been read in step i, and let $K(i)$ be the configuration (i.e., the complete stack, state, and input position) in step i. Since the stack size is bounded, $K(i)$ can be described with a finite number of symbols.

Now let v'' be the unique word with $v'v'' = v$. Having $K(i)$, v'' can be reconstructed with only finitely many additional symbols by using condition (i). The same holds true for the complete stack and the state after processing uv: In order to reconstruct them, we just start with the configuration $K(i)$ but take the previously reconstructed v'' as input. It follows that any word w which satisfies (ii) can be constructed with only c additional symbols. Because $K(i)$'s stack size is bounded by c_0, it follows that $C(w) \leq c'$ for some other constant $c' = c_0 + O(1)$.

Case 2. Now we show that Case 1 always holds. We assume that for infinitely many pairs (u, v) which satisfy (i) and (iii) the stack size will always be larger than c_0 during processing of the v part of the input.

Because of (i), for each u there exist only finitely many appropriate v. From the assumption that there are infinitely many such pairs (u, v) it follows then that we can choose u to be arbitrarily long and still find a v such that the stack size remains larger than c_0 while processing v on input uv. Using Proposition 2, we choose $|u|$ large enough such that there exists a number $r \in \mathbb{N}$ with $\lg(r) > |u|$ and

$$C(\underline{r}) < \lg(\lg(|u|)).$$

Using Proposition 4, we can find $s \leq c_0$ such that, after processing the suffix u' of $\cdots yyx$ with length $|u'| = r + s$, the dpda A will be in the same state as it has after processing u. In addition, the segment of the stack above position c_0 after processing u will also be the topmost segment after processing u'. Because the stack doesn't shrink below c_0, the observable behavior during processing v on input uv is the same as on input $u'v$. Hence, (i) implies that v can be reconstructed with finitely many additional symbols when only u' is known. Because s is bounded, we have $C(u') \leq C(\underline{r}) + O(1)$. For sufficiently large u we get

$$C(u') < \lg(\lg(|u|)).$$

Putting it all together, we get

$$\begin{aligned} C(v) &\leq C(u') + O(1) \\ &\leq \lg(\lg(|u|)) + O(1). \end{aligned}$$

This contradicts condition (iii). □

Since the KC-DCF lemma itself is difficult to apply, we also give the corresponding corrected formulation of its corollary (see [LiVi95]). We say that a program p *decides* L_1 *for* L_2 iff on input $u \in L_2$ the program p terminates and decides $u \in L_1$ correctly.

COROLLARY 6. *Let $L \subseteq \Sigma^*$ be a deterministic context-free language, $x, y \in \Sigma^*$, and let c be a constant. Then there exists a constant c' with the following property: If $u, v, w \in \Sigma^*$, where u is the suffix of $\cdots yyx$, then the following three conditions imply $C(w) \leq c'$:*

(i) *$C(v''|p_{uv'}) \leq c$ for all splittings $v'v'' = v$ and for all programs $p_{uv'}$ that decide $(uv')^{-1}L$ for* prefix$\{v''\}$;
(ii) *$C(w|p_{uv}) \leq c$ for all programs p_{uv} that list $(uv)^{-1}L$ in lexicographic order;*
(iii) *$C(v) \geq 2 \cdot \lg(\lg(|u|))$.*

Proof. Let A be a dpda that recognizes L. If $L = \emptyset$ or $L = \Sigma^*$, nothing has to be shown. Otherwise the KC-DCF lemma (Proposition 5) holds, and we can show that under the given premises, (i) and (ii) from the corollary imply (i) and (ii) from the KC-DCF lemma.

Assume that, under the given premises, u, v satisfy (i) from the corollary. Let $v'v'' = v$, and let k, q be the required additional information from Proposition 5(i). Now, if by using only finitely many additional symbols we can construct a program $p_{uv'}$ that decides $(uv')^{-1}L$ for prefix$\{v''\}$, then the bound in condition (i) from the corollary induces the bound in Proposition 5(i). This construction of $p_{uv'}$ can be obtained by observing the dpda A on input v'' when starting with state q and stack k.

With a similar argument, (ii) from above implies Proposition 5(ii). □

Though the original KC-DCF lemma and its corollary have been weakened in order to make them correct, they still can be applied for several nondeterministic context-free languages without much additional effort. We next give two examples taken from [LiVi95].

EXAMPLE 2. *The set of palindromes* $L = \{x \in \Sigma^* : x = x^{-1}\}$ *is not deterministic context-free.*

Proof. Set $y = 0$, $x = 1$, $u = 0^n1$, and $v = 0^n$. For an incompressible n we have

$$C(v) + O(1) = C(\underline{n}) \geq \lg(n) = \lg(|u| - 1),$$

implying Corollary 6(iii) for sufficiently large n. Let $v'v'' = v$; then v'' is the lexicographically first word in $(uv')^{-1}L$. Since all v are all 0, (i) is also satisfied. The lexicographically first word in $(uv)^{-1}L$ starting with 1 is 10^n; hence $w = 10^n$ satisfies (ii). On the other hand,

$$C(w) + O(1) = C(\underline{n}) \geq \lg(n),$$

and hence L cannot be deterministic context-free. □

EXAMPLE 3. $L = \{xy \in \{0,1\}^* : |x| = |y|$, *y contains at least one* $1\}$ *is not deterministic context-free.*

Proof. Set $y = 0$, $x = 1$, $u = 0^n1$, with $|u|$ even, and $v = 0^{n+1}$. For incompressible n we have

$$C(v) + O(1) = C(\underline{n}) \geq \lg(n) = \lg(|u| - 1),$$

thus implying Corollary 6(iii) for sufficiently large n. Let $v'v'' = v$. We need only one bit to code the information, whether $|v'|$ is even or odd. If $|v'|$ is even (odd), then v'' is the shortest word of even (odd) length that is all 0 and does not belong to $(uv')^{-1}L$. This shows that (i) is also satisfied. The lexicographically first word which does not belong to $(uv)^{-1}L$ and which starts with 1 is 10^{2n+3}. Thus with $w = 10^{2n+3}$, condition (ii) is satisfied, too. On the other hand, we have

$$C(w) + O(1) = C(\underline{n}) \geq \lg(n),$$

and hence L cannot be deterministic context-free. □

Acknowledgments. I'm indebted to the anonymous referee and to my colleagues Ulrike Brandt and Hermann K.-G. Walter for their valuable suggestions and corrections. Also I want to thank Ming Li for his kind email in response to my questions in 1999. Finally it should be mentioned that the idea behind the corrected KC-DCF lemma is essentially the same as that from the original paper by Ming Li and Paul Vitányi.

REFERENCES

[Ha78] M. A. Harrison, *Introduction to Formal Language Theory*, Addison–Wesley, Reading, MA, 1978.

[LiVi95] M. Li and P. Vitányi, *A new approach to formal language theory by Kolmogorov complexity*, SIAM J. Comput., 24 (1995), pp. 398–410.

[LiVi97] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd ed., Springer-Verlag, New York, 1997.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1395–1402

FINDING A PATH OF SUPERLOGARITHMIC LENGTH*

ANDREAS BJÖRKLUND† AND THORE HUSFELDT†

Abstract. We consider the problem of finding a long, simple path in an undirected graph. We present a polynomial-time algorithm that finds a path of length $\Omega\big((\log L/\log\log L)^2\big)$, where L denotes the length of the longest simple path in the graph. This establishes the performance ratio $O\big(n(\log\log n/\log n)^2\big)$ for the longest path problem, where n denotes the number of vertices in the graph.

Key words. approximation algorithms, graph algorithms, longest path

AMS subject classifications. 68R10, 68W25

DOI. 10.1137/S0097539702416761

1. Introduction.

Given an unweighted, undirected graph $G = (V, E)$ with $n = |V|$, the *longest path* problem is to find the longest sequence of distinct vertices $v_1 \cdots v_k$ such that $v_i v_{i+1} \in E$.

This is a classical optimization problem; the Hamiltonian path problem is a special case of it and appears in Karp's original list of NP-complete problems [8]. While today the approximability of most of these problems is well understood, the longest path problem has remained elusive, and notoriously so [5]: In spite of a considerable body of research the gap between upper and lower bounds is very wide.

Previous work. The first approximation algorithms for longest path are due to Monien [9] and Bodlaender [3], both finding a path of length $\Omega(\log L/\log\log L)$ in a graph with longest path length L.

A natural and harder variant of the problem is to find a path of length $\log n$ if it exists. Papadimitriou and Yannakakis [10] conjectured that this could be done in polynomial time, which was confirmed by Alon, Yuster, and Zwick [1], introducing the important method of *color-coding*. If the longest path has length $O(\log n)$, then their algorithm finds it (or, rather, it finds *a* longest path); else it finds a path of length $\Omega(\log n)$. Especially, the algorithm finds an $\Omega(\log L)$-path and thus has the performance ratio $O\big(n/\log n\big)$, which is the best ratio for the longest path problem known prior to the present paper.

Motivated by the weakness of these bounds for general graphs the problem has received additional study for restricted classes of graphs. In Hamiltonian graphs the algorithm of Vishwanathan [11] finds a path of length $O\big((\log n/\log\log n)^2\big)$. In *sparse* Hamiltonian graphs, Feder, Motwani, and Subi [5] find even longer paths. Moreover, they prove the following remarkable result: If a graph with vertices of degree at most 3 has a cycle of length r, then one can find in polynomial time a cycle of length at least r^c, where $c = \frac{1}{2}\log_3 2$.

The hardness results for this problem are mainly due to Karger, Motwani, and Ramkumar [7]: The longest path problem does not belong to APX unless P = NP, and it cannot be approximated within $2^{\log^{1-\epsilon} n}$ unless NP $\subseteq$ DTIME$\big(2^{O(\log^{1/\epsilon} n)}\big)$ for

*Received by the editors October 29, 2002; accepted for publication (in revised form) June 20, 2003; published electronically September 9, 2003. A preliminary version of this work was announced at the 29th International Colloquium on Automata, Languages, and Programming (ICALP) 2002.
http://www.siam.org/journals/sicomp/32-6/41676.html
†Department of Computer Science, Lund University, P.O. Box 118, SE–221 00 Lund, Sweden (thore@cs.lu.se).

any $\epsilon > 0$. More recently, it was shown that for *directed* graphs, the problem admits stronger lower bounds [2].

This paper. We present a polynomial-time algorithm that finds a path of length $\Omega\big((\log L/\log\log L)^2\big)$ in a graph with longest path length L. Since $L < n = |V|$, this corresponds to a performance ratio of order

$$O\left(\frac{n(\log\log n)^2}{\log^2 n}\right). \tag{1.1}$$

The main idea of our algorithm is a new graph decomposition which forms the basis of a recursive procedure. We find a cycle C of length $\log n/\log\log n$, using the algorithm from [3], remove C, and continue recursively in the resulting connected components. This decomposes the graph into a number of disjoint cycles of sufficient length which can be assembled into a long path.

The performance ratio (1.1) was obtained earlier by Vishwanathan [11] but only for Hamiltonian graphs.

For bounded degree graphs, we can improve the ratio to $O\big(n\log\log n/\log^2 n\big)$. For 3-connected graphs, we establish the performance ratio (1.1) for the *longest cycle* problem, a variant of the problem that also requires $v_1v_k \in E$.

2. Paths and cycles. In what follows, we consider a connected graph $G = (V, E)$ with $n = |V|$ vertices and $e = |E|$ edges. We write $G[W]$ for the graph induced by the vertex set W.

The *length* of a path and a cycle is its number of edges. The length of a cycle C is denoted $l(C)$. A k-cycle is a cycle of length k, and a k^+-cycle is a cycle of length k or larger. A k-path and k^+-path are defined similarly. For vertices x and y, an xy-path is a (simple) path from x to y. If P is a path containing u and v, we write $P[u, v]$ for the subpath from u to v. We let $L_G(v)$ denote the length of the longest path from a vertex v in the graph G. The *path length* of G is $\max_{v\in V} L_G(v)$.

We need the following result: Theorem 5.3(i) of [3].

THEOREM 1 (Bodlaender). *Given a graph, two of its vertices s, t, and an integer k, one can find a k^+-path from s to t (if it exists) in time $O\big((2k)!2^{2k}n + e\big)$.*

COROLLARY 1. *Given a graph, one of its vertices s, and an integer k, one can find a k^+-cycle through s (if it exists) in time $O\big(((2k)!2^{2k}n + e)n\big)$.*

Proof. For all neighbors t of s, apply the theorem on the graph with the edge st removed. □

We also need the following easy lemma.

LEMMA 1. *If a connected graph contains a path of length r, then every vertex is an endpoint of a path of length at least $\frac{1}{2}r$.*

Proof. Given vertices $u, v \in V$, let $d(u, v)$ denote the length of the shortest path between u and v.

Let $P = p_0 \cdots p_r$ be a path, and let v be a vertex. Find i minimizing $d(p_i, v)$. By minimality there is a path Q from v to p_i that contains no other vertices from P. Now either $QP[p_i, p_r]$ or $QP[p_i, p_0]$ has length at least $\frac{1}{2}r$. □

2.1. Decomposition into cycles. The next lemma is central to our construction and describes the graph decomposition that underlies our recursive algorithm. It formalizes the following observation: Assume that a vertex v originates a long path P and v lies on a cycle C. Then the removal of C decomposes G into connected components, one of which must contain a large part of P.

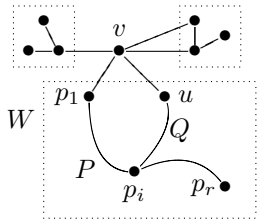


FIG. 1. *Statement* 1 *of Lemma* 2. *The path* $P = vp_1 \cdots p_r$ *continues in the component* W. *We assume that* v *does not lie on a large cycle. This means that an arbitrary path* Q *from* v*'s neighbor* u *must intersect* P *"early"; i.e.,* $QP[p_i, p_r]$ *is long.*

Pretending for a moment that our algorithm knew which component this is, we could continue the decomposition in it, recursively removing cycles until P is exhausted. In the end, we would have produced a long string of connected cycles. Especially, this string contains a path (using at least half the vertices of each cycle) that will be longer than the length of each individual cycle. The gist of this is that if we can find long *cycles* in graphs (like with Bodlaender's algorithm), then with our decomposition we can find even longer *paths*.

The lemma needs to distinguish between two cases, depending on whether or not v lies on a large cycle.

LEMMA 2. *Assume that a connected graph* G *contains a simple path* P *of length* $L_G(v) > 1$ *originating in vertex* v. *Then there exists a connected component* $G[W]$ *of* $G[V - v]$ *such that the following holds:*

1. *If* $G[W + v]$ *contains no* k^+*-cycle through* v, *then every neighbor* $u \in W$ *of* v *is the endpoint of a path of length*

$$L_{G[W]}(u) \geq L_G(v) - k.$$

2. *If* C *is a cycle in* $G[W + v]$ *through* v *of length* $l(C) < L_{G[W+v]}(v)$, *then there exists a connected component* H *of* $G[W - C]$ *that contains a neighbor* u *of* $C - v$ *in* $G[W + v]$. *Moreover, every such neighbor* u *is the endpoint of a path in* H *of length*

$$L_H(u) \geq \frac{L_G(v)}{2l(C)} - 1.$$

Proof. Let $r = L_G(v)$ and $P = p_0 \cdots p_r$, where $p_0 = v$. Note that $P[p_1, p_r]$ lies entirely in one of the components $G[W]$ of $G[V - v]$.

First consider statement 1; see Figure 1. Let $u \in W$ be a neighbor of v. Since $G[W]$ is connected, there exists a path Q from u to some vertex of P. Consider such a path. The first vertex p_i of P encountered on Q must have $i < k$, since otherwise the three paths vu, $Q[u, p_i]$, and $P[p_0, p_i]$ form a k^+-cycle. Thus the path $Q[u, p_i]P[p_i, p_r]$ has length at least $r - k + 1 > r - k$.

We proceed to statement 2; see Figure 2. Consider any cycle C in $G[W + v]$ through v. We will show that depending on how often P intersects C, there exists a long subpath in one of the components of $G[W - C]$. The length of this subpath is inversely proportional to the number of intersections, which could be no more than the length of C.

Case 1. First assume that $P \cap C = v$ so that one component H of $G[W - C]$ contains all of P except v. Let N be the set of neighbors of $C - v$ in H. First note

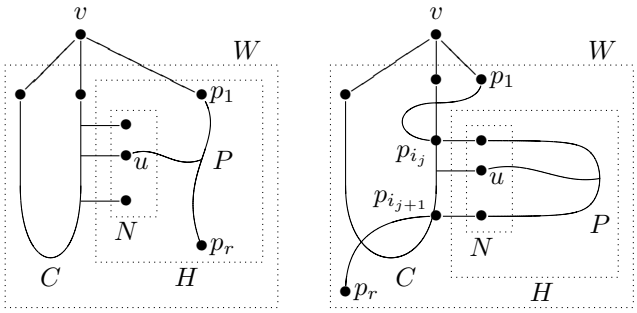


FIG. 2. *Statement* 2 *of Lemma* 2. *Here we assume that* v *does lie on a large cycle* C. *In Case* 1 *(left) the path* $P = vp_1 \cdots p_r$ *does not intersect* C *after it leaves* v. *Thus* $P[p_1, p_r]$ *lies entirely in a component* H *of* $W - C$. *Any neighbor* $u \in N$ *of* C *in this component must be the head of a long path using at least half of* $P[p_1, p_r]$. *In Case* 2 *(right) the path* P *intersects* C *in several places. Consider the largest section of* P *that lies entirely in a component* H *of* $W - C$, *here shown as a "loop" starting after* p_{i_j} *and ending before* $p_{i_{j+1}}$. *Any neighbor* $u \in N$ *of* C *in this component must be the head of a long path using at least half of the "loop."*

that N is nonempty, since $G[W]$ is connected. Furthermore, the path length of H is at least $r - 1$, so Lemma 1 gives $L_H(u) \geq (r-1)/2$ for every $u \in N$.

Case 2. Assume instead that $|P \cap C| = s > 1$. Enumerate the vertices on P from 0 to r and let $i_1, \ldots, i_s$ denote the indices of vertices in $P \cap C$, in particular $i_1 = 0$. Let $i_{s+1} = r$. An averaging argument shows that there exists j such that $i_{j+1} - i_j \geq r/s$. Consequently, there exists a connected component H of $G[W - C]$ containing a simple path of length $r/s - 2$. At least one of the i_jth or i_{j+1}th vertices of P must belong to $C - v$, so the set of neighbors N of $C - v$ in H must be nonempty. As before, Lemma 1 ensures $L_H(u) \geq r/2s - 1$ for every $u \in N$, which establishes the bound after noting that $s \leq l(C)$. □

3. Result and algorithm. The construction in this section and its analysis establishes the following theorem, accounting for the worst-case performance ratio of (1.1) as claimed in the introduction.

THEOREM 2. *If a graph contains a simple path of length* L, *then we can find a simple path of length*

$$\Omega\left(\left(\frac{\log L}{\log\log L}\right)^2\right)$$

in polynomial time.

We first give a brief overview of the algorithm; the next two sections will provide the details.

Assume for simplicity that the input graph is connected; if not, then we can iterate the algorithm over each connected component of the input graph and return the longest path found.

Pick any vertex v. Lemma 1 ensures that v is the head of a path of length at least $r > L/2$. In the next sections we will pretend that we know the value

$$k = \left\lceil \frac{2\log r}{\log\log r} \right\rceil;$$

but this is not a restriction since we can (in polynomial time) run the algorithm for every value of $k = 6, \ldots, \lceil 2\log n/\log\log n \rceil$ and return the longest path found.

Given v and k we will construct a tree $T_k(G, v)$ as detailed in section 3.1; this tree will describe a recursive decomposition of the input graph G into paths and cycles. Finally, we find a long (weighted) path in $T_k(G, v)$. This path will describe a path in G which will have the desired length as shown in section 3.2.

In summary, assuming a connected input graph, the algorithm proceeds as follows:

1. Pick any vertex $v \in G$.
2. For every $k = 6, \ldots, \lceil 2 \log n / \log \log n \rceil$ perform the following two steps and return the longest path found:
3. Construct the tree $T_k(G, v)$ as detailed in section 3.1.
4. Find a longest weighted path in $T_k(G, v)$ and return the path in G described by it, as detailed in section 3.2.

Steps 3 and 4 take polynomial time (see below), so the entire algorithm takes polynomial time.

3.1. Construction of the cycle decomposition tree. Given a vertex v in G, our algorithm constructs a node-weighted tree $T_k = T_k(G, v)$, rooted at v, called the *cycle decomposition tree*. Every node of T_k is either a *singleton* or a *cycle* node: A singleton node corresponds to a single vertex $u \in G$ and is denoted $\langle u \rangle$, while a cycle node corresponds to a cycle C with a specified vertex $u \in C$ and is denoted $\langle C, u \rangle$. Every singleton node has unit weight, and every cycle node $\langle C, u \rangle$ has weight $\frac{1}{2} l(C)$.

The tree $T_k(G, v)$ is constructed as follows. Initially, T_k contains a singleton node $\langle v \rangle$, and a call is made to the following procedure with arguments G and v:

1. [Iterate over components:] For every maximal connected component $G[W]$ of $G[V - v]$, execute step 2.
2. [Find cycle:] Search for a k^+-cycle through v in $G[W + v]$ using Theorem 1. If such a cycle C is found, then execute step 3; otherwise, execute step 5.
3. [Insert cycle node:] Insert the cycle node $\langle C, v \rangle$ and the tree edge $\langle v \rangle \langle C, v \rangle$. For every connected component H of $G[W - C]$ execute step 4.
4. [Recurse:] Choose an arbitrary neighbor $u \in H$ of $C - v$, and insert the singleton node $\langle u \rangle$ and the tree edge $\langle u \rangle \langle C, v \rangle$. Then, recursively execute step 1 to compute $T_k(H, u)$.
5. [Insert singleton node and recurse:] Pick an arbitrary neighbor $u \in G[W + v]$ of v, insert the node $\langle u \rangle$ and the tree edge $\langle v \rangle \langle u \rangle$, and recursively execute step 1 to compute $T_k\big(G[W], u\big)$.

Note that each recursive step constructs a tree that is connected to other trees by a single edge, so T_k is indeed a tree. Also note that the ancestor of every cycle node must be a singleton node. The root of T_k is $\langle v \rangle$.

To see that the running time of this procedure is polynomial, first note that step 2 is polynomial because of the corollary to Theorem 1. The number of recursive steps is linear, since every step inserts a node into T_k, which is clearly of linear size after the procedure.

3.2. Paths in the cycle decomposition tree. Our algorithm proceeds by finding a path of greatest weight in T_k. This can be done in linear time by depth first search. The path found in T_k represents a path in G if we interpret paths through cycle nodes as follows. Consider a path in T_k through a cycle node $\langle C, u \rangle$. Both neighbors are singleton nodes, so we consider the subpath $\langle u \rangle \langle C, u \rangle \langle v \rangle$. By construction, v is connected to some vertex $w \in C$ with $w \neq u$. One of the two paths from u to w in C must have length at least half the length of C; call it P. We will interpret the path $\langle u \rangle \langle C, u \rangle \langle v \rangle$ in T_k as a path uPv in G. If a path ends in a cycle node $\langle C, u \rangle$, we may associate it with a path of length $l(C) - 1$ by moving along C from u in any of its two

directions. Thus a path of weight m in T_k from the root to a leaf identifies a path of length at least m in G.

We need to show that T_k for some small k has a path of sufficient length.[1]

LEMMA 3. *If G contains a path of length $r \geq 2^8$ starting in v, then $T_k = T_k(G, v)$ for*

$$k = \left\lceil \frac{2 \log r}{\log \log r} \right\rceil$$

contains a weighted path of length at least $\frac{1}{8}k^2 - \frac{1}{4}k - 1$.

Proof. We follow the construction of T_k in section 3.1.

We need some additional notation. For a node $x = \langle w \rangle$ or $x = \langle C, w \rangle$ in T_k we let $L(x)$ denote the length of the longest path from w in the component $G[X]$ corresponding to the subtree rooted at x. More precisely, for every successor y of x (including $y = x$), the set X contains the corresponding vertices w' (if $y = \langle w' \rangle$ is a singleton node) or C' (if $y = \langle w', C' \rangle$ is a cycle node).

Furthermore, let $\mathbf{S}(n)$ denote the singleton node children of a node n, and let $\mathbf{C}(n)$ denote its cycle node children. Consider any singleton node $\langle v \rangle$.

Lemma 2 asserts that

$$L(v) \leq \max \left\{ \max_{w \in \mathbf{S}\langle v \rangle} L(w) + k, \max_{\substack{\langle C, v \rangle \in \mathbf{C}\langle v \rangle \\ w \in \mathbf{S}\langle C, v \rangle}} \big(2L(w) + 2\big) l(C) \right\}. \tag{3.1}$$

Define $n(v) = w$ if $\langle w \rangle$ maximizes the right-hand side of the inequality (3.1), and consider a path $Q = \langle x_0 \rangle \cdots \langle x_t \rangle$ from $\langle v \rangle = \langle x_0 \rangle$ described by these heavy nodes. To be precise, we have either $n(x_i) = x_{i+1}$ or $n(x_i) = x_{i+2}$; in the latter case, the predecessor of $\langle x_{i+2} \rangle$ is a cycle node.

We will argue that the gaps in the sequence

$$L(x_0) \geq L(x_1) \geq \cdots \geq L(x_t)$$

cannot be too large due to the inequality above. This, combined with the fact that $L(x_t)$ must be small (otherwise, we are done), implies that Q contains a lot of cycle nodes or even more singleton nodes.

Let s denote the number of cycle nodes on Q. Since every cycle node has weight at least $\frac{1}{2}k$ the total weight of Q is at least $\frac{1}{2}sk + (t - s) = s(\frac{1}{2}k - 1) + t$.

Consider a singleton node that is followed by a cycle node. There are s such nodes; we will call them *cycle parents*. Assume $\langle x_j \rangle$ is the first cycle parent node. Thus, according to the first part of Lemma 2, its predecessors $\langle x_0 \rangle, \ldots, \langle x_j \rangle$ satisfy the relation $L(x_{i+1}) \geq L(x_i) - k$, so

$$L(x_j) \geq r - jk \geq r - \tfrac{1}{8}k^3 \geq \tfrac{7}{8}r,$$

since $j \leq t \leq \frac{1}{8}k^2$ (otherwise, we are finished) and $r \geq k^3$.

From the second part of Lemma 2 we have

$$L(x_{j+2}) \geq \frac{7r}{16 l(C)} - 1 \geq \frac{r}{k^2},$$

[1] All logarithms are to the base 2, and the constants involved have been chosen aiming for simplicity of the proof rather than optimality.

where we have used $l(C) \le \frac{1}{4}k^2$ (otherwise, we are finished) and $r \ge \frac{4}{3}k^2$.

This analysis may be repeated for the subsequent cycle parents as long as their remaining length after each cycle node passage is at least k^3. Note that Q must pass through as many as $s' \ge \lceil \frac{1}{4}k - 1 \rceil$ cycle nodes before

$$\frac{r}{k^{2s'}} < k^3,$$

at which point the remaining path may be shorter than k^3. Thus we either have visited $s \ge s'$ cycle nodes, amounting to a weighted path Q of length at least

$$s(\tfrac{1}{2}k + 1) \ge \tfrac{1}{8}k^2 - \tfrac{1}{4}k - 1$$

(remembering that any two consecutive cycle nodes must have a singleton node in between), or there are at most $s < s'$ cycle nodes on Q. In that case there is a tail of singleton nodes starting with some $L(x) \ge k^3$. Since $L(x_j) \le L(x_{j+1}) + k$ for the nodes on the tail, the length of the tail (and thus the weight of Q) is at least k^2. □

It remains to check that the path found by our algorithm satisfies the stated approximation bound: For the right k, the preceding lemma guarantees a weighted path in $T_k(G, v)$, and hence a path in G, of length

$$\frac{k^2}{8} - \frac{k}{4} - 1 = \Omega\left(\left(\frac{\log r}{\log\log r}\right)^2\right) = \Omega\left(\left(\frac{\log L}{\log\log L}\right)^2\right)$$

because $r \ge \frac{1}{2}L$ by Lemma 1. This finishes the proof of Theorem 2.

4. Extensions.

4.1. Bounded degree graphs. As in [11], the class of graphs with their maximum degree bounded by a constant admits a relative $\log\log n$-improvement over the performance ratio shown in this paper. All paths of length $\log n$ can be enumerated in polynomial time for these graphs. Consequently, we can replace the algorithm from Theorem 1 by an algorithm that efficiently finds cycles of logarithmic length or larger through any given vertex if they exist.

PROPOSITION 1. *If a constant degree graph contains a simple path of length L, then we can find a simple path of length*

$$\Omega\left(\frac{\log^2 L}{\log\log L}\right)$$

in polynomial time.

This gives the performance ratio $O\big(n \log\log n / \log^2 n\big)$ for the longest path problem in constant degree graphs.

4.2. 3-connected graphs. Bondy and Locke [4] have shown that every 3-connected graph with path length L must contain a cycle of length at least $2L/5$. Moreover, their construction is easily seen to be algorithmic and efficient. This implies the following result on the longest cycle problem.

PROPOSITION 2. *If a 3-connected graph contains a simple cycle of length L, then we can find a simple cycle of length*

$$\Omega\left(\left(\frac{\log L}{\log\log L}\right)^2\right)$$

in polynomial time.

This gives the performance ratio $O\big(n(\log\log n/\log n)^2\big)$ for the longest cycle problem in 3-connected graphs. Note that for 3-connected cubic graphs, [5] shows a considerably better bound.

Acknowledgments. We thank Andrzej Lingas for bringing [11] to our attention and Gerth Stølting Brodal for commenting on a previous version of this paper.

Note added in proof. Recently, Gabow and Nie [6] have improved the bound in Corollary 1 to $O(e) + 2^{O(k)} n \log n$. As a consequence, the bounds in Theorem 2 and Proposition 2 are improved to $\Omega(\log^2 L/\log\log L)$, and the performance ratio for longest path becomes $O\big(|V|\log\log|V|/\log^2|V|\big)$.

REFERENCES

[1] N. Alon, R. Yuster, and U. Zwick, *Color-coding*, J. ACM, 42 (1995), pp. 844–856.
[2] A. Björklund, T. Husfeldt, and S. Khanna, *Approximating Longest Directed Path*, Technical report TR03-032, Electronic Colloquium on Computational Complexity, Vol. 10, 2003.
[3] H. L. Bodlaender, *On linear time minor tests with depth-first search*, J. Algorithms, 14 (1993), pp. 1–23.
[4] J. A. Bondy and S. C. Locke, *Relative length of paths and cycles in 3-connected graphs*, Discrete Math., 33 (1981), pp. 111–122.
[5] T. Feder, R. Motwani, and C. Subi, *Approximating the longest cycle problem in sparse graphs*, SIAM J. Comput., 31 (2002), pp. 1596–1607.
[6] H. N. Gabow and S. Nie, *Finding a Long Directed Cycle*, CU Technical report CU-CS-961-03, University of Colorado, Boulder, CO, 2003.
[7] D. Karger, R. Motwani, and G. D. S. Ramkumar, *On approximating the longest path in a graph*, Algorithmica, 18 (1997), pp. 82–98.
[8] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–103.
[9] B. Monien, *How to find long paths efficiently*, Ann. Discrete Math., 25 (1985), pp. 239–254.
[10] C. H. Papadimitriou and M. Yannakakis, *On limited nondeterminism and the complexity of the V–C dimension*, J. Comput. System Sci., 53 (1996), pp. 161–170.
[11] S. Vishwanathan, *An approximation algorithm for finding a long path in Hamiltonian graphs*, in Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2000, pp. 680–685.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1403–1422

BETTER ALGORITHMS FOR UNFAIR METRICAL TASK SYSTEMS AND APPLICATIONS*

AMOS FIAT† AND MANOR MENDEL†

Abstract. Unfair metrical task systems are a generalization of online metrical task systems. In this paper we introduce new techniques to combine algorithms for unfair metrical task systems and apply these techniques to obtain improved randomized online algorithms for metrical task systems on arbitrary metric spaces.

Key words. online algorithms, randomized algorithms

AMS subject classifications. 68W20, 68W25, 68W40

DOI. 10.1137/S0097539700376159

1. Introduction. Metrical task systems (MTSs), introduced by Borodin, Linial, and Saks [11], can be described as follows: A server in some internal state receives *tasks* that have a service cost associated with each of the internal states. The server may switch states, paying a cost given by a metric space defined on the state space, and then pays the service cost associated with the new state.

MTSs have been the subject of a great deal of study. A large part of the research into online algorithms can be viewed as a study of some particular MTS. In modelling some of these problems as MTSs, the set of permissible tasks is constrained to fit the particulars of the problem. In this paper we consider the original definition of MTSs, where the set of tasks can be arbitrary.

A deterministic algorithm for any n-state MTS with a competitive ratio of $2n-1$ is given in [11], along with a matching lower bound for any metric space.

The *randomized* competitive ratio of the MTS problem is not as well understood. For the uniform metric space, where all distances are equal, the randomized competitive ratio is known to within a constant factor and is $\Theta(\log n)$ [11, 16]. In fact, it has been conjectured that the randomized competitive ratio for MTS is $\Theta(\log n)$ in any n-point metric space. Previously, the best upper bound on the competitive ratio for arbitrary n-point metric space was $O(\log^5 n \log\log n)$ due to Bartal et al. [3] and Bartal [2]. The best lower bound for arbitrary n-point metric space is $\Omega(\log n/\log\log n)$ due to Bartal, Bollobás, and Mendel [4] and Bartal et al. [5], improving the previous lower bounds of Karloff, Rabani, and Ravid [18] and Blum et al. [10].

As observed in [18, 10, 1], the randomized competitive ratio of the MTS is conceptually easier to analyze on "decomposable spaces": spaces that have a partition to subspaces with a small diameter compared to that of the entire space. Bartal [1] introduced a class of decomposable spaces called *hierarchically well-separated trees* (HSTs). Informally, a k-HST is a metric space having a partition into subspaces such that (i) the distances between the subspaces are all equal; (ii) the diameter of each

*Received by the editors August 2, 2000; accepted for publication (in revised form) May 15, 2003; published electronically September 9, 2003. This work was partly supported by United States Israel Bi-National Science Foundation grant 96-00247/1. A preliminary version of this paper appeared in the *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, Portland, OR, 2000, pp. 725–734.

http://www.siam.org/journals/sicomp/32-6/37615.html

†School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel (fiat@tau.ac.il, mendelma@yahoo.com).

subspace is at most $1/k$ times the diameter of the whole space; and (iii) each subspace is recursively a k-HST.

Following [1, 3], we obtain an improved algorithm for HSTs. In order to reduce the MTS problem on arbitrary metric space to an MTS problem on a HST we use probabilistic embedding of metric spaces into HSTs [1]. It is shown in [2] that any n-point metric space has probabilistic embedding in k-HSTs with *distortion* $O(k \log n \log\log n)$. Thus, an MTS problem on an arbitrary n-point metric space can be reduced to an MTS problem on a k-HST with overhead of $O(k \log n \log\log n)$ [1].

Our algorithm for HSTs follows the general framework given in [10] and explicitly formulated in [20, 3], where the recursive structure of the HST is modelled by defining an unfair metrical task system (UMTS) problem [20, 3] on a uniform metric space. In a UMTS problem, associated with every point v_i of the metric space is a cost ratio r_i. We charge the online algorithm a cost of $r_i c_i$ for dealing with the task $(c_1, \ldots, c_i, \ldots, c_n)$ in state v_i, which multiplies the online costs for processing tasks in that point. Offline costs remain as before. The cost ratio r_i roughly corresponds to the competitive ratio of the online algorithm in a subspace of the HST. For UMTSs on uniform metric spaces, tight upper bounds are known only for two point spaces [10, 20, 3] and for n-point spaces with equal cost ratios [3]. A tight lower bound is known for any number of points and any cost ratios [4].

In this paper we introduce a general notation and technique for combining algorithms for UMTSs on hierarchically decomposable metric spaces. This technique is an improvement on the previous methods [10, 20, 3]. Using this technique, we obtain randomized algorithms for UMTSs on the uniform metric space that are better than the algorithm of [3]. Using the algorithm for UMTSs on uniform metric space and the new method for combining algorithms, we obtain $O(\log n \log\log n)$-competitive algorithms for MTS on HST spaces, which implies an $O((\log n \log\log n)^2)$-competitive randomized algorithm for MTSs on any metric space.

We also study the *weighted caching problem*. Weighted caching is the paging problem where there are different costs to fetch different pages. Deterministically, a competitive ratio of k is achievable [12, 23], with a matching lower bound following from the k-server bound [19]. No randomized algorithm is known to have a competitive ratio better than the deterministic competitive ratio for general metric spaces. However, in some special cases progress has been made. Irani [15] has shown an $O(\log k)$-competitive algorithm when page fetch costs are one of two possible values. Blum, Furst, and Tomkins [9] have given an $O(\log^2 k)$-competitive algorithm for arbitrary page costs when the total number of pages is $k+1$; they also present a lower bound of $\Omega(\log k)$ for any page costs. As the weighted caching problem on $k+1$ pages with cache size k is a special case of MTS on star-like metric spaces, we are able to obtain an $O(\log k)$-competitive algorithm for this case, improving [9]. This is tight up to a constant factor.

Outline of the paper. In section 2 the MTS problem is formally defined, along with several technical conditions that later allow us to combine algorithms for subspaces together. In section 3 we deal with the main technical contribution of our paper. We introduce a novel technique to combine algorithms for subspace into an algorithm for the entire space. Section 4 is devoted to introducing algorithms for UMTSs on uniform spaces. In section 5 we give the applications mentioned above by combining the algorithms of section 4.

2. Preliminaries. UMTSs [20, 3] are a generalization of MTSs [11]. A UMTS $U = (M; (r_u)_{u \in M}; s)$ consists of a metric space M with a distance metric d_M, a

sequence of *cost ratios* $r_u \in \mathbb{R}^+$ for $u \in M$, and a *distance ratio* $s \in \mathbb{R}^+$.

Given a UMTS U, the associated online problem is defined as follows. An online algorithm A occupies some state $u \in M$. When a task arrives the algorithm may change state to v. A task is a tuple $(c_x)_{x \in M}$ of nonnegative real numbers, and the cost for algorithm A associated with servicing the task is $s \cdot \mathrm{d}_M(u,v) + r_v c_v$. The cost for A associated with servicing a sequence of tasks σ is the sum of costs for servicing the individual tasks of the sequence consecutively. We denote this sum by $\mathrm{cost}_A(\sigma)$. An online algorithm makes its decisions based only upon tasks seen so far.

An offline player is defined that services the same sequence of tasks over U. The cost of an offline player, if it were to do exactly as above, would be $\mathrm{d}_M(u,v) + c_v$. Thus, the concept of *unfairness*, the costs for doing the same thing, are different.

Given a sequence of tasks σ we define the *work function* [13] at v, $w_{\sigma,U}(v)$, to be the minimal cost, for any offline player, to start at the initial state in U, deal with all tasks in σ, and end up in state v. We omit the use of the subscript U if it is clear from the context. Note that for all $u, v \in M$, $w_\sigma(u) - w_\sigma(v) \leq \mathrm{d}_M(u,v)$. If $w_\sigma(u) = w_\sigma(v) + \mathrm{d}_M(u,v)$, u is said to be *supported* by v. We say that $u \in M$ is *supported* if there exists some $v \in M$ such that u is supported by v.

We define $\mathrm{cost}_{\mathrm{OPT}}(\sigma)$ to be $\min_v w_\sigma(v)$. This is simply the minimal cost, for any offline player, to start at the initial state and process σ. As the differences between the work function values on different states is bounded by a constant (the diameter of the metric space) independent of the task sequence, it is possible to use a convex combination of the work function values instead of the minimal one. We say that $\alpha = (\alpha(u))_{u \in M}$ is a *weight vector* when $\{\alpha(u) | u \in M\}$ are nonnegative real numbers satisfying $\sum_{u \in M} \alpha(u) = 1$. We define the α-optimal-cost of a sequence of tasks σ to be $\mathrm{cost}_{\alpha\text{-OPT}}(\sigma) = \langle \alpha, w_\sigma \rangle = \sum_{u \in M} \alpha(u) w_\sigma(u)$. As observed above, $\mathrm{cost}_{\alpha\text{-OPT}}(\sigma) \leq \mathrm{cost}_{\mathrm{OPT}}(\sigma) + \mathrm{diam}(M)$, where $\mathrm{diam}(M) = \max_{u,v \in M} \mathrm{d}_M(u,v)$ is the diameter of M.

A randomized online algorithm A for a UMTS is a probability distribution over deterministic online algorithms. The expected cost of a randomized algorithm A on a sequence σ is denoted by $E[\mathrm{cost}_A(\sigma)]$.

DEFINITION 2.1 (see [22, 17, 7]). *A randomized online algorithm A is called r-competitive against an oblivious adversary if there exists some c such that for all task sequences σ, $E[\mathrm{cost}_A(\sigma)] \leq r\ \mathrm{cost}_{\mathrm{OPT}}(\sigma) + c$.*

Observation 1. We can limit the discussion on the competitive ratio of UMTSs to distance ratios equal to one since a UMTS $U = (M; (r_u)_{u \in M}; s)$ has a competitive ratio of r if and only if $U' = (M; (s^{-1} r_u)_{u \in M}; 1)$ has a competitive ratio of rs^{-1}. Moreover, an rs^{-1} competitive algorithm for U' is an r-competitive algorithm for U, since in both U' and U the offline costs are the same, but the online costs in U are multiplied by a factor of s compared to the costs in U'. When $s = 1$, we drop it from the notation.

Given a randomized online algorithm A for a UMTS U with state space M and a sequence of tasks σ, we define $p_{\sigma,A}$ to be the vector of probabilities $(p_{\sigma,A}(u))_{u \in M}$, where $p_{\sigma,A}(u)$ is the probability that A is in state u after serving the request sequence σ. We drop the subscript A if the algorithm is clear from the context.

Let $x \circ y$ denote the concatenation of sequences x and y. Let U be a UMTS over the metric space M with distance ratio s. Given two successive probability distributions on the states of U, p_σ and $p_{\sigma \circ e}$, where e is the next task, we define the set of transfer matrices from p_σ to $p_{\sigma \circ e}$, denoted $T(p_\sigma, p_{\sigma \circ e})$, as the set of all matrices

$T = (t_{uv})_{u,v \in M}$ with nonnegative real entries, where

$$\sum_{v \in M} t_{uv} = p_\sigma(u),\ u \in M; \qquad \sum_{u \in M} t_{uv} = p_{\sigma \circ e}(v),\ v \in M.$$

We define the *unweighted moving cost* from p_σ to $p_{\sigma \circ e}$:

$$\mathrm{mcost}_M(p_\sigma, p_{\sigma \circ e}) = \min_{\substack{(t_{uv}) \in \\ T(p_\sigma, p_{\sigma \circ e})}} \sum_{u,v} t_{uv}\, \mathrm{d}_M(u,v),$$

the *moving cost* is defined as $\mathrm{mcost}_U(p_\sigma, p_{\sigma \circ e}) = s \cdot \mathrm{mcost}_M(p_\sigma, p_{\sigma \circ e})$, and the *local cost* on a task $e = (c_u)_{u \in M}$ is defined as $\sum_{u \in M} p_{\sigma \circ e}(u) c_u r_u$. Due to linearity of expectation, $E[\mathrm{cost}_A(\sigma \circ e)] - E[\mathrm{cost}_A(\sigma)]$ is equal to the sum of the moving cost from p_σ to $p_{\sigma \circ e}$ and the local cost on e. Hence we can view A as a deterministic algorithm that maintains the probability mass on the states whose cost on task e given after sequence σ is

$$\mathrm{cost}_A(\sigma \circ e) - \mathrm{cost}_A(\sigma) = \mathrm{mcost}_U(p_\sigma, p_{\sigma \circ e}) + \sum_{u \in M} p_{\sigma \circ e}(u) c_u r_u. \tag{2.1}$$

In what follows we will use the terminology of changing probabilities, with the understanding that we are referring to a deterministic algorithm charged according to (2.1).

We next develop some technical conditions that make it easier to combine algorithms for UMTSs. *Elementary tasks* are tasks with only one nonzero entry; we use the notation (v, δ), $\delta \geq 0$, for an elementary task of cost δ at state v. Tasks $(v, 0)$ can simply be ignored by the algorithm.

DEFINITION 2.2 (see [3]). *A* reasonable *algorithm is an online algorithm that never assigns a positive probability to a supported state.*

DEFINITION 2.3 (see [3]). *A* reasonable task sequence *for algorithm A is a sequence of tasks that obeys the following:*

1. *All tasks are elementary.*
2. *For all σ, the next task (v, δ) must obey that for all δ', if $\delta > \delta' \geq 0$, then $p_{\sigma \circ (v, \delta')}(v) > 0$.*

It follows that a reasonable task sequence for A never includes tasks (v, δ), $\delta > 0$, if the current probability of A on v is zero.

The following lemma is from [3]. For the sake of completeness, we include a sketch of a proof here.

LEMMA 2.4. *Given a randomized online algorithm A_0 that obtains a competitive ratio of r when the task sequences are limited to being reasonable task sequences for A_0, then, for all $\varepsilon > 0$, there also exists a randomized algorithm A_3 that obtains a competitive ratio of $r + \varepsilon$ on all possible sequences.*

Proof (sketch). The proof proceeds in three stages. In the first stage, we convert an algorithm A_0 for reasonable task sequences to a *lazy* algorithm A_1 (an algorithm that does not move the server when receiving a task with zero cost) for reasonable task sequences. In the second stage, we convert an algorithm A_1 to an algorithm A_2 for elementary task sequences, and then, in the third stage, we convert A_2 to an algorithm A_3 for general task sequences.

The first stage is well known.

The second stage. Given an elementary task sequence, every elementary task $e = (v, x)$ is converted to a task (v, y) such that $y = \sup\{z | z < x$ and the probability

induced by A_1 on v is greater than 0}. The resulting task sequence is reasonable and is fed to A_1. A_2 imitates the movements of A_1.

The third stage. Let σ be an arbitrary task sequence. First, we convert σ into an elementary task sequence $\hat{\sigma}$; each task $\tau = (\delta_1, \ldots, \delta_n)$ in σ is converted to a sequence of tasks $\hat{\sigma}_\tau$ as follows: Let $\varepsilon' > 0$ be a small constant to be determined later, and assume for simplicity that $\delta_i \geq \delta_{i+1}$. Then $\hat{\sigma}_\tau = \varsigma_1 \circ \varsigma_2 \circ \cdots \circ \varsigma_N$, where $N = \lfloor \delta_1/\varepsilon' \rfloor$ and $\varsigma_j = (v_1, \varepsilon') \circ (v_2, \varepsilon') \circ \cdots \circ (v_{k_j}, \varepsilon')$, where $k_j = \max\{i | \delta_i \geq j \cdot \varepsilon'\}$. Note that the optimal offline cost on $\hat{\sigma}$ is at most the optimal offline cost on σ, since any servicing for σ when applied to $\hat{\sigma}$ would have a cost no bigger than the original cost. Consider an r-competitive online algorithm A_2 for elementary tasks operating on $\hat{\sigma}$, and construct an online algorithm A_3 for σ. A_3 maintains the invariant that the state of A_3 after processing some task τ is the same state as A_2 after processing the sequence $\hat{\sigma}_\tau$. Consider the behavior of A_2 on $\hat{\sigma}_\tau$. It begins in some state v_{i_0}, passes through some set S of states, and ends up in some state v_{i_2}. Consider the original task $\tau = (\delta_1, \ldots, \delta_n)$. Let v_{i_1} be the state in S with the lowest cost in τ. Algorithm A_3 begins in state v_{i_0}, immediately moves to v_{i_1}, serves τ in v_{i_1}, and then moves to v_{i_2}.

Informally, on each task A_2 pays either a local cost of ε' or a moving cost of at least ε', and therefore these costs are larger than the local cost of A_3. A_3 also has a moving cost of at most the moving cost of A_2. By a careful combination of these two we conclude that the cost of A_3 on σ is at most $(1 + \varepsilon)$ times the cost of A_2 on $\hat{\sigma}$. □

Hereafter, we assume only reasonable task sequences. This is without loss of generality due to Lemma 2.4.

Observation 2. When a reasonable algorithm A is applied to a reasonable task sequence $\sigma = \tau_1 \tau_2 \cdots \tau_m$, any elementary task $\tau = (v, \delta)$ causes the work function at v, $w(v)$, to increase by δ. This follows because v would not have been supported following any alternative request (v, δ'), $\delta' < \delta$. See [3, Lemma 1] for a rigorous treatment. This also implies that for any state v, $w_\sigma(v) = \sum_{j=1}^{m} \tau_j(v)$.

DEFINITION 2.5. *An online algorithm A is said to be* sensible *and r-competitive on the UMTS $U = (M; (r_u)_{u \in M}; s)$ if it obeys the following:*

1. *A is reasonable.*
2. *A is a* stable algorithm [13], *i.e., the probabilities that A assigns to the different states are purely a function of the work function.*
3. *Associated with A are a weight vector α_A and a potential function Φ_A such that*
   - *$\Phi_A : \mathbb{R}^b \mapsto \mathbb{R}^+$ is purely a function of the work function, bounded, nonnegative, and continuous.*
   - *For all task sequences σ and all tasks e,*

$$\text{cost}_A(\sigma \circ e) - \text{cost}_A(\sigma) + \Phi_A(w_{\sigma \circ e}) - \Phi_A(w_\sigma) \leq r \cdot \langle \alpha_A, w_{\sigma \circ e} - w_\sigma \rangle. \tag{2.2}$$

Observation 3. An online algorithm that is sensible and r-competitive (against reasonable task sequences) according to Definition 2.5 is also r-competitive according to Definition 2.1. This is so since summing up the two sides in inequality (2.2) over the individual tasks in the task sequence, we get a telescopic sum such that $\text{cost}_A(\sigma) + \Phi_A(w_\sigma) - \Phi_A(w_\varepsilon) \leq r \cdot \langle \alpha_A, w_\sigma - w_\varepsilon \rangle$, where w_ε is the initial work function. We conclude that $\text{cost}_A(\sigma) \leq r \cdot \text{cost}_{\text{OPT}}(\sigma) + r\Delta(M) + \sup_w \Phi(w)$.

When combining sensible algorithms we would like the resulting algorithm to also be sensible. The problematic invariant to maintain is reasonableness. In order

to maintain reasonableness there is a need for a stronger concept, which we call *constrained algorithms*.

DEFINITION 2.6. *A sensible r-competitive algorithm A for the UMTS $U = (M; (r_u)_{u\in M}; s)$ with associated potential function Φ is called (β, η)-constrained, $0 \le \beta \le 1$, $0 \le \eta$, if the following hold:*

1. *For all $u, v \in M$, if $w(u) - w(v) \ge \beta\, \mathrm{d}_M(u, v)$, then the probability that A assigns to u is zero ($p_{w,A}(u) = 0$).*
2. *$\|\Phi\|_\infty \le \eta\ \mathrm{diam}(M) r$, where $\|\Phi\|_\infty = \sup_w \Phi(w)$.*

Observation 4.

1. For a (β, η)-constrained algorithm competing against a reasonable task sequence, *for all* $u, v \in M$, $|w(u) - w(v)| \le \beta\ \mathrm{d}_M(u, v)$. The argument here is similar to the one given in Observation 2.
2. A sensible r-competitive algorithm for a metric space of diameter Δ is by definition $(1, |\Phi_A|/(r\Delta))$-constrained.
3. A (β, η)-constrained algorithm is trivially (β', η')-constrained for all $\beta \le \beta' \le 1$ and $\eta \le \eta'$.

3. A combining theorem for UMTSs. Consider a metric space M having a partition to subspaces $M_1, \ldots, M_b$, with "large" distances between subspaces compared to the diameters of the subspaces. A MTS on M induces MTSs on M_i, $i \in \{1, \ldots, b\}$. Assume that for every i, we have an $\hat{r}_i$-competitive algorithm A_i for the induced MTS on M_i. Our goal is to combine the A_i algorithms so as to obtain an algorithm for the original MTS defined on M. To do so we make use of a "combining algorithm" $\hat{A}$. $\hat{A}$ has the role of determining which of the M_i subspaces contains the server. Since the "local cost" of $\hat{A}$ on subspace M_i is $\hat{r}_i$ times the optimal cost on subspace M_i, it is natural that $\hat{A}$ should be an algorithm for the UMTS $\hat{U} = (\hat{M}; (\hat{r}_1, \ldots, \hat{r}_b))$, where $\hat{M} = \{z_1, \ldots, z_b\}$ is a space with points corresponding to the subspaces and distances that are roughly the distances between the corresponding subspaces. Tasks for M are translated to tasks for the M_i induced MTSs simply by restriction. It remains to define how one translates tasks for M to tasks for $\hat{U}$.

Previous papers [10, 20, 3] use the cost of the optimal algorithm for the task in the subspace M_i as the cost for z_i in the task for $\hat{U}$. This way the local cost for $\hat{A}$ is $\hat{r}_i$ times the cost for the optimum; however, *this is true only in the amortized sense*. In order to bound the fluctuation around the amortized cost, those papers have to assume that the diameters of the subspace are very small compared to the distances between M_i subspaces. We take a different approach: the cost for a point $z_i \in \hat{U}$ is (an upper bound for) *the cost of A_i on the corresponding task, divided by $\hat{r}_i$*. In this way the amortization problem disappears, and we are able to combine subspaces with a relatively large diameter. A formal description of the construction is given below.

THEOREM 3.1. *Let U be a UMTS $U = (M; (r_u)_{u\in M}; s)$, where M is a metric space on n points. Consider a partition of the points of M, $P = (M_1, M_2, \ldots, M_b)$. $U_j = (M_j; (r_u)_{u\in M_j}; s)$ is the UMTS induced by U on the subspace M_j. Let $\hat{M}$ be a metric space defined over the set of points $\{z_1, z_2, \ldots, z_b\}$ with a distance metric $\mathrm{d}_{\hat{M}}(z_i, z_j) \ge \max\{\mathrm{d}_M(u, v) : u \in M_i, v \in M_j\}$. Assume that*

- *for all j, there is a (β_j, η_j)-constrained $\hat{r}_j$-competitive algorithm A_j for the UMTS U_j;*
- *there is a $(\hat{\beta}, \hat{\eta})$-constrained r-competitive algorithm $\hat{A}$ for the UMTS $\hat{U} = (\hat{M}; (\hat{r}_1, \ldots, \hat{r}_b); s)$.*

Define

(3.1)
$$\beta = \max\left\{ \max_i \beta_i,\ \max_{i\neq j} \frac{\hat{\beta}\,\mathrm{d}_{\hat{M}}(z_i,z_j) + \beta_j\ \mathrm{diam}(M_j) + \beta_i\ \mathrm{diam}(M_i) + \eta_i\,\mathrm{diam}(M_i)}{\min_{p\in M_i, q\in M_j} \mathrm{d}_M(p,q)} \right\}$$

and

$$\eta = \hat{\eta}\frac{\mathrm{diam}(\hat{M})}{\mathrm{diam}(M)} + \max_i\ \eta_i \frac{\mathrm{diam}(M_i)}{\mathrm{diam}(M)}. \tag{3.2}$$

If $\beta \leq 1$, then there exists a (β,η)-constrained and r-competitive algorithm, A, for the UMTS U.

In our applications of Theorem 3.1, the metric space M has a "nice" partition $P = (M_1,\ldots,M_b)$, parameterized with $k \geq 1$: $d_M(u,v) = \mathrm{diam}(M)$ for all $i \neq j$ $u \in M_i$, $v \in M_j$; and $\mathrm{diam}(M_i) \leq \mathrm{diam}(M)/k$. In this case the statement of Theorem 3.1 can be simplified as follows.

COROLLARY 3.2. *Under the assumptions of Theorem* 3.1*, and assuming the partition is "nice" (with parameter k), in the above sense, define*

$$\beta = \max\left\{ \max_i \beta_i, \hat{\beta} + \frac{\max_{i\neq j}(\beta_i+\beta_j+\eta_i)}{k} \right\} \tag{3.3}$$

and

$$\eta = \hat{\eta} + \frac{\max_i \eta_i}{k}. \tag{3.4}$$

If $\beta \leq 1$, then there exists a (β,η)-constrained and r-competitive algorithm, A, for the UMTS U.

In section 3.1 we define the combined algorithm A declared in Theorem 3.1. Section 3.2 contains the proof of Theorem 3.1. We end the discussion on the combining technique with section 3.3, in which we show how to obtain constrained algorithms needed in the assumptions of Theorem 3.1.

3.1. The construction of the combined algorithm. Denote by Φ_j and α_j the associated potential function and weight vector of algorithm A_j, respectively. Similarly, denote by $\hat{\Phi}$ and $\hat{\alpha}$ the associated potential function and weight vector of algorithm $\hat{A}$, respectively.

Given a sequence of elementary tasks $\sigma = (v_1,\delta_1)\circ(v_2,\delta_2)\circ\cdots\circ(v_{|\sigma|},\delta_{|\sigma|})$, $v_i \in M$, we define the sequences

$$\sigma|_{M_\ell} = (u_1^\ell,\delta_1^\ell)\circ(u_2^\ell,\delta_2^\ell)\circ\cdots\circ(u_{|\sigma|}^\ell,\delta_{|\sigma|}^\ell),\ \text{where}$$

- $u_j^\ell = v_j$ and $\delta_j^\ell = \delta_j$ if $v_j \in M_\ell$,
- u_j^ℓ is an arbitrary point in M_ℓ and $\delta_j^\ell = 0$ if $v_j \notin M_\ell$.

Informally, $\sigma|_{M_\ell}$ is the restriction of σ to subspace M_ℓ.

For $u \in M$, define $s(u) = i$ if and only if $u \in M_i$. We define the sequence

$$\chi(\sigma) = (z_{s(v_1)},\hat{\delta}_1)\circ(z_{s(v_2)},\hat{\delta}_2)\circ\cdots\circ(z_{s(v_{|\sigma|})},\hat{\delta}_{|\sigma|})$$

inductively. Let $e = (v,\delta)$, $s(v) = \ell$, then $\chi(\sigma\circ e) = \chi(\sigma)\circ(z_\ell,\hat{\delta})$, where

(3.5)
$$\hat{\delta} = \big(\langle \alpha_\ell, w_{(\sigma\circ e)|_{M_\ell},U_\ell}\rangle - \Phi_\ell(w_{(\sigma\circ e)|_{M_\ell},U_\ell})/\hat{r}_\ell\big) - \big(\langle \alpha_\ell, w_{\sigma|_{M_\ell},U_\ell}\rangle - \Phi_\ell(w_{\sigma|_{M_\ell},U_\ell})/\hat{r}_\ell\big).$$

Note that $\hat{\delta}$ is an upper bound on the cost of A_ℓ for the task (v, δ), divided by $\hat{r}_\ell$. This fact follows from (2.2) since A_ℓ is sensible, and $\sigma|_{M_\ell}$ is a reasonable task sequence for A_ℓ (see Lemma 3.3). It also implies that $\hat{\delta} \geq 0$, which is a necessary requirement for $(z_\ell, \hat{\delta})$ to be a well-defined task.

ALGORITHM A. The algorithm works as follows:

1. It simulates algorithm A_ℓ on the task sequence $\sigma|_{M_\ell}$ for $1 \leq \ell \leq b$.
2. It also simulates algorithm $\hat{A}$ on the task sequence $\chi(\sigma)$.
3. The probability assigned to a point $v \in M_\ell$ is the product of the probability assigned by A_ℓ to v and the probability assigned by $\hat{A}$ to z_ℓ (i.e., $p_{\sigma,A}(v) = p_{\sigma|_{M_\ell},A_\ell}(v) \cdot p_{\chi(\sigma),\hat{A}}(z_\ell)$).

We remark that the simulations above can be performed in an online fashion.

3.2. Proof of Theorem 3.1. To simplify notation we use the following shorthand notation. Given a task sequence σ and a task e, with respect to σ, we define

$$\begin{aligned} w &= w_{\sigma,U}; & w^e &= w_{\sigma\circ e,U}; \\ w_k &= w_{\sigma|_{M_k},U_k},\ 1 \leq k \leq b; & w_k^e &= w_{(\sigma\circ e)|_{M_k},U_k},\ 1 \leq k \leq b; \\ \hat{w} &= w_{\chi(\sigma),\hat{U}}; & \hat{w}^e &= w_{\chi(\sigma\circ e),\hat{U}}. \end{aligned}$$

Define p, p_k, and $\hat{p}$ to be the probability distributions on the states of U, U_k, and $\hat{U}$ as induced by algorithms A, A_k, and $\hat{A}$ on the sequences σ, $\sigma|_{M_k}$, and $\chi(\sigma)$, $1 \leq k \leq b$, respectively. Likewise, we define p^e, p_k^e, and $\hat{p}^e$, where the sequences are $\sigma \circ e$, $\sigma \circ e|_{M_k}$, and $\chi(\sigma \circ e)$.

LEMMA 3.3. *If the task sequence σ given to algorithm A on U is reasonable, then the simulated task sequences $\sigma|_{M_i}$ for algorithms A_i on U_i and the simulated task sequence $\chi(\sigma)$ for algorithm $\hat{A}$ on $\hat{U}$ are also reasonable.*

Proof. We first prove that $\sigma'|_{M_\ell}$ is reasonable for A_ℓ by induction on $|\sigma'|$. Say $\sigma' = \sigma \circ e$, $e = (v, \delta)$, and $v \in M_\ell$. Since σ' is reasonable for A, if the task e would have been replaced with the task $e' = (v, \delta')$, for $\delta' \in [0, \delta)$, then by the reasonableness of σ', $p^{e'}(v) > 0$. Since $p^{e'}(v) = p_\ell^{e'}(v)\hat{p}^{e'}(z_\ell)$ it follows that $p_\ell^{e'}(v) > 0$. This implies $\sigma'|_{M_\ell}$ is reasonable for A_ℓ.

We next prove that $\chi(\sigma')$ is a reasonable task sequence for $\hat{A}$, by induction on $|\sigma'|$. Let $\sigma' = \sigma \circ e$, $e = (v, \delta)$, $v \in M_\ell$. Denote by $\hat{e} = (z_\ell, \hat{\delta})$ the last task in $\chi(\sigma)$. Consider a hypothetical task (v, x) in U for $0 \leq x \leq \delta$. Denote by $(z_\ell, f(x))$ the corresponding task for $\hat{U}$, where $f(x)$ is determined according to (3.5). f is continuous (since Φ_ℓ is continuous), $f(0) = 0$, and $f(\delta) = \hat{\delta}$. Therefore for any $0 \leq \hat{\delta}' < \hat{\delta}$ there exists $0 \leq \delta' < \delta$ such that $f(\delta') = \hat{\delta}'$, and since $0 < p^{(v,\delta')} = p_\ell^{(v,\delta')}(v) \cdot \hat{p}^{(v,\delta')}(z_\ell)$ we conclude that $0 < \hat{p}^{(v,\delta')}(z_\ell)$ (the probability induced by $\hat{A}$ on z_ℓ after the task $(z_\ell, \hat{\delta}')$). This implies that $\chi(\sigma)$ is a reasonable task sequence for $\hat{A}$. ∎

LEMMA 3.4. *For all σ and for all ℓ, $\hat{w}(z_\ell) = \langle \alpha_\ell, w_\ell \rangle - \Phi_\ell(w_\ell)/\hat{r}_\ell$.*

Proof. It follows from Lemma 3.3 that the task sequence $\chi(\sigma)$ for $\hat{A}$ is reasonable. As $\hat{A}$ is sensible it follows from Observation 2 that $\hat{w}(z_\ell)$ is exactly the sum of costs in $\chi(\sigma)$ for z_ℓ. By the definition of $\chi(\sigma)$ (see (3.5)) it follows that this sum is $\langle \alpha_\ell, w_\ell \rangle - \Phi_\ell(w_\ell)/\hat{r}_\ell$. ∎

LEMMA 3.5. *Assume that $w(u) = w_\ell(u)$ for all $1 \leq \ell \leq b$, $u \in M_\ell$. Then any state $u \in U$ for which there exists a state v such that $w(u) - w(v) \geq \beta\, \mathrm{d}_M(u, v)$ has $p(u) = 0$.*

Proof. Consider states u and v as above, i.e., $w(u) - w(v) \geq \beta\, \mathrm{d}_M(u, v)$. We now consider two cases:

1. $u, v \in M_i$. We want to show that $w_i(u) - w_i(v) \geq \beta_i \, \mathrm{d}_{M_i}(u, v)$, as A_i is (β_i, η_i)-constrained; this implies that $p_i(u) = 0$, which implies that $p(u) = 0$. From the conditions above we get
$$w_i(u) - w_i(v) = w(u) - w(v) \geq \beta \, \mathrm{d}_M(u, v) \geq \beta_i \, \mathrm{d}_{M_i}(u, v).$$
2. $u \in M_i$, $v \in M_j$, $i \neq j$. Our goal now will be to show that $\hat{w}(z_i) - \hat{w}(z_j) \geq \hat{\beta} \, \mathrm{d}_{\hat{M}}(z_i, z_j)$, as this implies that $\hat{p}(z_i) = 0$, which implies that $p(u) = 0$.
A lower bound on $\hat{w}(z_i)$ is
$$\hat{w}(z_i) = \langle \alpha_i, w_i \rangle - \|\Phi_i\|_\infty / r_i \tag{3.6}$$
$$\geq w_i(u) - \beta_i \operatorname{diam}(M_i) - \|\Phi_i\|_\infty / r_i \tag{3.7}$$
$$= w(u) - \beta_i \operatorname{diam}(M_i) - \eta_i \operatorname{diam}(M_i). \tag{3.8}$$
To justify (3.6) one uses the definitions and Lemma 3.4. Inequality (3.7) follows because a convex combination of values is at least one of these values minus the maximal difference. The maximal difference between work function values is bounded by β_i times the distance; see Observation 4. Equation (3.8) follows from our assumption that the work functions are equal and from the definition of η_i.
Similarly, to obtain an upper bound on $\hat{w}(z_j)$, we derive
$$\hat{w}(z_j) = \langle \alpha_j, w_j \rangle - \|\Phi_j\|_\infty / r_j \leq w(v) + \beta_j \operatorname{diam}(M_j). \tag{3.9}$$
It follows from (3.8) and (3.9) that
$$\hat{w}(z_i) - \hat{w}(z_j) \geq (w(u) - w(v)) - \beta_i \operatorname{diam}(M_i) - \beta_j \operatorname{diam}(M_j) - \eta_i \operatorname{diam} M_i$$
$$\geq \beta d_M(u, v) - \beta_i \operatorname{diam}(M_i) - \beta_j \operatorname{diam}(M_j) - \eta_i \operatorname{diam} M_i \geq \hat{\beta} \, \mathrm{d}_{\hat{M}}(z_i, z_j).$$
The last inequality follows from (3.1). □

LEMMA 3.6. *For any reasonable task sequence σ, subspace M_ℓ, and $v \in M_\ell$ it holds that $w_\ell(v) = w(v)$.*

Proof. Assume the contrary. Let σ' be the shortest reasonable task sequence for which there exists $v \in M_\ell$ satisfying $w_{\sigma'|_{M_\ell}}(v) \neq w_{\sigma'}(v)$. It is easy to observe that $\sigma' = \sigma \circ e$, where $e = (v, \delta)$. As the sequence $(\sigma \circ e)|_{M_\ell}$ is a reasonable task sequence (Lemma 3.3) and A_ℓ is reasonable, it follows that $w_\ell^e(v) = w_\ell(v) + \delta$. Since $w_\ell(v) = w(v)$ and $w^e(v) \leq w(v) + \delta$ we deduce that $w_\ell^e(v) > w^e(v)$.

Let $e_x = (v, x)$; define $\delta' = \sup\{x : w^{e_x}(v) = w_\ell^{e_x}(v)\}$. Obviously, $0 \leq \delta' \leq \delta$. Define $e' = (v, \delta')$. By continuity of the work function, $w^{e'}(v) = w_\ell^{e'}(v)$, and thus $\delta' < \delta$. The conditions above imply that an elementary task in v after $w^{e'}$ will not change the work function, which means that v is supported in $w^{e'}$. Hence, the assumptions of Lemma 3.5 are satisfied (here we use the assumption that $\beta \leq 1$). By Lemma 3.5, $p^{e'}(v) = 0$, and since the sequence σ is reasonable for A, it follows that $\delta \leq \delta'$, a contradiction. □

PROPOSITION 3.7. *For all σ and all tasks $e = (v, \delta)$,*
$$\operatorname{cost}_A(\sigma \circ e) - \operatorname{cost}_A(\sigma) \leq \operatorname{cost}_{\hat{A}}(\chi(\sigma \circ e)) - \operatorname{cost}_{\hat{A}}(\chi(\sigma)).$$

Proof. Let us denote the subspace containing v by M_ℓ. We split the cost of A into two main components: the moving cost $\operatorname{mcost}_U(p, p^e)$ and the local cost $r_v p^e(v)\delta = r_v \hat{p}^e(z_\ell) p_\ell(v_i)\delta$ (see (2.1)).

We give an upper bound on the moving cost of A by considering a possibly suboptimal algorithm that works as follows:

1. Move probabilities between the different M_j subspaces; i.e., change the probability $p(u) = \hat{p}(z_j)p_j(u)$ for $u \in M_j$ to an intermediate stage $\hat{p}^e(z_j)p_j(v)$. The moving cost for A to produce this intermediate probability is bounded by $\mathrm{mcost}_{\hat{U}}(\hat{p}, \hat{p}^e)$, as the distances in $\hat{M}$ are an upper bound on the real distances for A ($\mathrm{d}_{\hat{M}}(z_i, z_j) \geq \mathrm{d}_M(u, v)$ for $u \in M_i$, $v \in M_j$). We call this cost the interspace cost for A.
2. Move probabilities within the M_j subspaces; i.e., move from the intermediate probability $\hat{p}^e(z_j)p_j(u)$, $u \in M_j$, to the probability $p^e(u) = \hat{p}^e(z_j)p^e_j(u)$. As all algorithms A_j, $j \neq \ell$, get a task of zero cost, $p^e_j = p_j$, $j \neq \ell$. The moving cost for A to produce $p^e(u)$, $u \in M_\ell$, from the intermediate stage, is no more than $\hat{p}^e(z_\ell) \cdot \mathrm{mcost}_{U_\ell}(p_\ell, p^e_\ell)$. We call this cost the intraspace cost for A.

Taking the local cost for A and the intraspace cost for A,

$$
\begin{aligned}
&r_u \hat{p}^e(z_\ell)p_\ell(u)\delta + \hat{p}^e(z_\ell) \cdot \mathrm{mcost}_{U_\ell}(p_\ell, p^e_\ell) \\
&\qquad = \hat{p}^e(z_\ell)\left(\mathrm{cost}_{A_\ell}(\sigma \circ e) - \mathrm{cost}_{A_\ell}(\sigma)\right) && (3.10) \\
&\qquad \leq \hat{p}^e(z_\ell)\hat{r}_\ell\big((\langle \alpha_\ell, w^e_\ell\rangle - \Phi_\ell(w^e_\ell)/\hat{r}_\ell) - (\langle \alpha_\ell, w_\ell\rangle - \Phi_\ell(w_\ell)/\hat{r}_\ell)\big). && (3.11)
\end{aligned}
$$

To obtain (3.10) we use the definition of online cost (see (2.1)). To obtain (3.11) we use the fact that A_ℓ is $\hat{r}_\ell$-competitive and sensible (see (2.2)).

Let $\hat{e}$ be the last task in $\chi(\sigma \circ e)$. Formula (3.11) is simply the local cost for algorithm $\hat{A}$ on task $\hat{e}$. Thus, we have bounded the cost for algorithm A on task e to be no more than the cost for algorithm $\hat{A}$ on task $\hat{e}$. □

Proof of Theorem 3.1. We associate a weight vector α and a bounded potential function Φ with algorithm A, where

$$
\alpha(v) = \hat{\alpha}(z_\ell)\alpha_\ell(v) \quad \text{for } v \in M_\ell\,; \qquad \Phi(w) = \hat{\Phi}(\hat{w}) + r\sum_i \hat{\alpha}(z_i)\Phi_i(w_i)/\hat{r}_i.
$$

We remark that from Lemma 3.4 and Lemma 3.6 it follows that $\hat{w}$ and w_i are determined by w, so $\Phi(w)$ is well defined.

We derive the following upper bound on the cost of A:

$$
\begin{aligned}
&\mathrm{cost}_A(\sigma \circ e) - \mathrm{cost}_A(\sigma) \\
&\qquad \leq \mathrm{cost}_{\hat{A}}(\chi(\sigma \circ e)) - \mathrm{cost}_{\hat{A}}(\chi(\sigma)) && (3.12) \\
&\qquad \leq r\Big(\sum_i \hat{\alpha}(z_i)\hat{w}^e(z_i) - \sum_i \hat{\alpha}(z_i)\hat{w}(z_i)\Big) - \Big(\hat{\Phi}(\hat{w}^e) - \hat{\Phi}(\hat{w})\Big) && (3.13) \\
&\qquad = r\left(\sum_i \sum_{v \in M_i} \hat{\alpha}(z_i)\alpha_i(v)w^e_i(v) - \sum_i \sum_{v \in M_i} \hat{\alpha}(z_i)\alpha_i(v)w_i(v)\right) \\
&\qquad\quad - \left(\left(\hat{\Phi}(\hat{w}^e) + r\sum_i \hat{\alpha}(z_i)\Phi_i(w^e_i)/\hat{r}_i\right) - \left(\hat{\Phi}(\hat{w}) + r\sum_i \hat{\alpha}(z_i)\Phi_i(w_i)/\hat{r}_i\right)\right) && (3.14) \\
&\qquad = r(\langle \alpha, w^e\rangle - \langle \alpha, w\rangle) - (\Phi(w^e) - \Phi(w)). && (3.15)
\end{aligned}
$$

Inequality (3.12) follows from Proposition 3.7. Inequality (3.13) is implied as $\hat{A}$ is a sensible r-competitive algorithm. We obtain (3.14) by substituting $\hat{w}^e(z_i)$ and $\hat{w}(z_i)$ according to Lemma 3.4 and rearranging the summands. Equation (3.15) follows from the definition of α and Φ above, and using Lemma 3.6.

We now prove that A is (β, η)-constrained. It follows from Lemma 3.5 and Lemma 3.6 that the condition on β is satisfied (see Definition 2.6). It remains to

show the condition on η:

$$\|\Phi\|_\infty \le \|\hat{\Phi}\|_\infty + r\sum_i \hat{\alpha}(z_i)\|\Phi_i\|_\infty/\hat{r}_i \tag{3.16}$$

$$\le \hat{\eta} r \cdot \operatorname{diam}(\hat{M}) + r\sum_i \hat{\alpha}(z_i)\eta_i\hat{r}_i \cdot \operatorname{diam}(M_i)/\hat{r}_i \tag{3.17}$$

$$\le r \cdot \operatorname{diam}(M)\big(\hat{\eta}\tfrac{\operatorname{diam}(\hat{M})}{\operatorname{diam}(M)} + \max_i\{\eta_i \tfrac{\operatorname{diam}(M_i)}{\operatorname{diam}(M)}\}\big)$$

$$= r \cdot \operatorname{diam}(M)\eta.$$

Inequality (3.16) follows by the definition of Φ; (3.17) follows because $\hat{A}$ is $(\hat{\beta},\hat{\eta})$-constrained and A_i is (β_i,η_i)-constrained, $1 \le i \le b$.

We have therefore shown that A is a (β,η)-constrained and r-competitive algorithm. ∎

3.3. Constrained algorithms. Theorem 3.1 assumes the existence of constrained algorithms. In this section we show how to obtain such algorithms. The proof is motivated by similar ideas from [20, 3].

DEFINITION 3.8. *Fix a metric space M on b states and cost ratios $r_1,\dots,r_b$. Assume that for all $s>0$ there is a (β,η)-constrained $f(s)$-competitive algorithm A_s for the UMTS $U_s = (M; r_1,\dots,r_b; s)$ against reasonable task sequences. For $\rho > 0$ we define the ρ-variant of A_s (if it exists) to be a $(\beta\rho,\eta\rho)$-constrained $f(s/\rho)$-competitive algorithm for U_s.*

LEMMA 3.9. *Let $0 < \beta \le 1$ and $0 < \beta/\rho \le 1$. Assume there exists a $(\beta/\rho,\eta/\rho)$-constrained and r-competitive online algorithm A' for the UMTS $U' = (\rho M; r_1,\dots,r_b; s/\rho)$. Then there exists a (β,η)-constrained and r-competitive algorithm A for the UMTS $U = (M; r_1,\dots,r_b; s)$.*

Proof. Algorithm A on the UMTS U simulates algorithm A' on the UMTS U' by translating every task (v,δ) to task (v',δ). The probability that A associates with state v is the same as the probability that algorithm A' associates with state v'. If the task sequence for A' is reasonable, then the simulated task sequence for A' is also reasonable simply because the probabilities for v and v' are identical.

The costs of A or A' on task (v,δ) or (v',δ) can be partitioned into moving costs and local costs. As the probability distributions are identical, the local costs for A and A' are the same. The unweighted moving costs for A are $1/\rho$ the unweighted moving costs for A' because all distances are multiplied by $1/\rho$. However, the moving costs for A' are the unweighted moving costs multiplied by a factor of s/ρ, whereas the moving costs for A are the unweighted moving costs multiplied by a factor of s. Thus, the moving costs are also equal.

To show that A is (β,η)-constrained (and hence reasonable) we first need to show that if the work functions in U and U' are equal, then this implies that if u and v are two states such that $w(u) \ge w(v) + \beta\,\mathrm{d}_M(u,v)$, then $p(u) = 0$. This is true because A' is $(\beta/\rho,\eta/\rho)$-constrained, and thus $w(u') \ge w(v') + (\beta/\rho)\cdot \mathrm{d}_{\rho M}(u',v')$ implies a probability of zero on u' for A', which implies a probability of zero on u for A. Next, one needs to show that the work functions are the same; this can be done using an argument similar to the proof of Lemma 3.6.

As the work functions and costs are the same for the online algorithms A and A' it follows that we can use the same potential function. To show that $|\Phi| \le \eta\cdot\operatorname{diam}(M)$ we note that $|\Phi| \le (\eta/\rho)\operatorname{diam}(\rho M)$. ∎

Observation 5. Assume there exists a (β,η)-constrained and r-competitive algorithm A for a UMTS $U=(M;r_1,\ldots,r_b;s)$. Then, for all $\rho>0$, a natural modification of A, A', is a (β,η)-constrained, r-competitive algorithm for the UMTS $U'=(\rho M;r_1,\ldots,r_b;s)$.

LEMMA 3.10. *Under the assumptions of Definition* 3.8*, for all* $\rho>0$ *such that* $\beta\rho\le 1$*, and for all* $s>0$*, the* ρ*-variant of* A_s *exists.*

Proof. For all $\rho>0$ such that $\beta\rho\le 1$, we have the following:

1. By the assumption, there exists a (β,η)-constrained, $f(s/\rho)$-competitive algorithm for the UMTS $(M;r_1,\ldots,r_b;s/\rho)$.
2. It follows from Lemma 3.9 that there exists an online algorithm that is $(\rho\beta,\rho\eta)$-constrained, $f(s/\rho)$-competitive for the UMTS $(\rho^{-1}M;r_1,\ldots,r_b;s)$.
3. It now follows from Observation 5 that there exists a $(\rho\beta,\rho\eta)$-constrained, $f(s/\rho)$-competitive online algorithm for the UMTS $(M;r_1,\ldots,r_b;s)$. This means that the ρ variant of A_s exists. ∎

4. The uniform metric space. Let $\mathcal{U}_b^d$ denote the metric space on b points, where all pairwise distances are d (a uniform metric space). In this section we develop algorithms for UMTSs whose underlying metric is uniform. We begin with two special cases that were previously studied in the literature.

The first algorithm works for the UMTS $U=(\mathcal{U}_b^d;(r_1,\ldots,r_b);s)$, $b\ge 2$, and $r_1=r_2=\cdots=r_b$. However, it can be defined for arbitrary cost ratios. The algorithm, called ODDEXPONENT, was defined and analyzed in [3]. Applying our terminology to the results of [3], we obtain the following lemma.

LEMMA 4.1. ODDEXPONENT *is* $(1,1)$*-constrained and* $(\max_i r_i+6s\ln b)$*-competitive.*

Proof. Algorithm ODDEXPONENT, when servicing a reasonable task sequence, allocates for configuration v the probability $p(v)=\frac{1}{b}+\frac{1}{b}\sum_u\left(\frac{w(u)-w(v)}{d}\right)^t$, where t is chosen to be an odd integer in the range $[\ln b,\ln b+2)$.

In our terminology, Bartal et al. [3] prove that ODDEXPONENT is sensible, that it is $(\max_i r_i+6s\ln b)$-competitive, and that the associated potential function $|\Phi_1|\le(\max_i r_i/(t+1)+s)d\le(1/\lceil\ln b\rceil)(\max_i r_i+6s\ln b)d$. This implies that ODDEXPONENT is $(1,1/\lceil\ln b\rceil)$-constrained. ∎

The second algorithm works for the two point UMTS $U=(\mathcal{U}_2^d;r_1,r_2;s)$. The algorithm, called TWOSTABLE, was defined and analyzed in [20] and [3], based on an implicit description of the algorithm that appeared previously in [10]. Applying our terminology to the results of [20, 3], we obtain the following lemma.

LEMMA 4.2. TWOSTABLE *is* $(1,4)$*-constrained and* r*-competitive, where*

$$r=r_1+\frac{r_1-r_2}{e^{(r_1-r_2)/s}-1}=r_2+\frac{r_2-r_1}{e^{(r_2-r_1)/s}-1}.$$

Proof. TWOSTABLE works as follows: Let $y=w(v_1)-w(v_2)$, and $z=(r_1-r_2)/s$. The probability on point v_1 is $p(v_1)=\left(e^z-e^{z(\frac{1}{2}+\frac{y}{2d})}\right)/\left(e^z-1\right)$. TWOSTABLE is shown to be sensible and r-competitive in [3, 20], and the potential function associated with TWOSTABLE, Φ_2, obeys $|\Phi_2|\le(2r_2+s)d$.

It remains to show that $|\Phi_2|\le 4rd$. We use the fact that, in general, if $|z|\le 1/2$, then $1/2\le z/(e^z-1)$, and we do a simple case analysis. If $\max\{r_1,r_2\}>\frac{1}{2}s$, then $|\Phi_2|\le(2r_2+s)d\le(2r+2r)d\le 4rd$. Otherwise, $|z|\le 1/2$, so $r=r_2+\frac{z}{e^z-1}s\ge r_2+\frac{s}{2}$. Hence $|\Phi_2|\le 2rd$. ∎

To gain an insight about the competitive ratio of TwoStable, we have the following proposition.

Proposition 4.3. *Let* $f(s,r_1,r_2) = r_1+(r_1-r_2)/\big(e^{(r_1-r_2)/s}-1\big)$. *Let* $x_1,x_2 \in \mathbb{R}^+$ *such that* $r_1 \le 2s(\ln x_1+1)$ *and* $r_2 \le 2s(\ln x_2+1)$. *Then* $f(s,r_1,r_2) \le 2s(\ln(x_1+x_2)+1)$.

Proof. First we show that f is a monotonic nondecreasing function of both r_1 and r_2. Since the formula is symmetric in r_1 and r_2 it is enough to check monotonicity in r_1. Let $x=(r_1-r_2)/s$; it suffices to show that $g(x)=sx+r_2+sx/(e^x-1)$ is monotonic in x. The derivative satisfies

$$g'(x) = s\cdot\frac{e^x(e^x-(1+x))}{(e^x-1)^2} \ge 0, \text{ since } e^x \ge 1+x.$$

Therefore we may assume that $r_1 = 2s(\ln x_1+1)$ and $r_2 = 2s(\ln x_2+1)$. Without loss of generality we can assume that $x_1 \ge x_2$, and let $y \ge 2$ be such that $x_1 = (x_1+x_2)(1-1/y)$. By substitution we get $r_1-r_2 = 2s\ln(y-1)$ and

$$\begin{aligned} f(s,r_1,r_2) = r_1+\frac{r_1-r_2}{e^{(r_1-r_2)/s}-1} &= 2s\Big(\ln(x_1+x_2)+1+\ln(y-1)-\ln y+\frac{\ln(y-1)}{(y-1)^2-1}\Big) \\ &\le 2s\Big(\ln(x_1+x_2)+1-\frac{1}{y}+\frac{\ln(y-1)}{(y-1)^2-1}\Big). \end{aligned}$$

We now prove that for $y \ge 2$, $-\frac{1}{y}+\frac{\ln(y-1)}{(y-1)^2-1} \le 0$. When y approaches 2, the limit of the expression is zero. For $y>2$, we multiply the left side by $(y-1)^2-1$ and get $g(y) = -(y-2)+\ln(y-1)$. Since $g(2)=0$ and $g'(y) = -1+1/(y-1) < 0$ for $y>2$, we are done. ∎

We next describe a new algorithm, called Combined, defined on a UMTS $U = (\mathcal{U}_b^d; r_1,\dots,r_b; s)$. This algorithm is inspired by *Strategy* 3 [3]. Like *Strategy* 3, Combined combines OddExponent and TwoStable on subspaces of $\mathcal{U}_b^d$; however, it does so in a more sophisticated way that is impossible using the combining technique of [3]. Figure 4.1 presents the scheme of the combining process.

Algorithm Combined. As discussed in Observation 1, we may assume that $s=1$. Let x_i be the minimal real number such that $r_i \le 100\ln x_i \ln\ln x_i$ and $x_i \ge e^{e^6+1}$, and let x denote $\sum_i x_i$. For a set $S \subset M_b^d$ let $U(S)$ denote the UMTS induced by U on S.

Let $\mathcal{U}_b^d = \{v_1,\dots,v_b\}$, where v_i has cost ratio r_i. We partition the points of $\mathcal{U}_b^d$ as follows: let $Q_\ell = \{v_i : e^{\ell-1} \le x_i < e^\ell\}$. Let $P = \{Q_\ell : |Q_\ell| \ge \ln x\} \cup \{\{v\} : v \in Q_\ell \text{ and } |Q_\ell| < \ln x\}$, P being a partition of $\mathcal{U}_b^d$. For $S \in P$ let $x(S) = \sum_{v_i \in S} x_i$. Without loss of generality we assume $P = \{S_1, S_2, \dots, S_{b'}\}$, where $b' = |P|$ and $x(S_j) \ge x(S_{j+1})$, $1 \le j \le b'-1$.

We associate with every set S_i an algorithm $A(S_i)$ on the UMTS $U(S_i)$. If $|S_i| \ge \ln x$ we choose $A(S_i)$ to be the (1/10)-variant of OddExponent. If $|S_i| < \ln x$, then $|S_i| = 1$, and we choose $A(S_i)$ to be the trivial algorithm on one point; this algorithm has a competitive ratio equal to the cost ratio, and it is $(0,0)$-constrained. Let $r(S_i)$ denote the competitive ratio of $A(S_i)$ on $U(S_i)$.

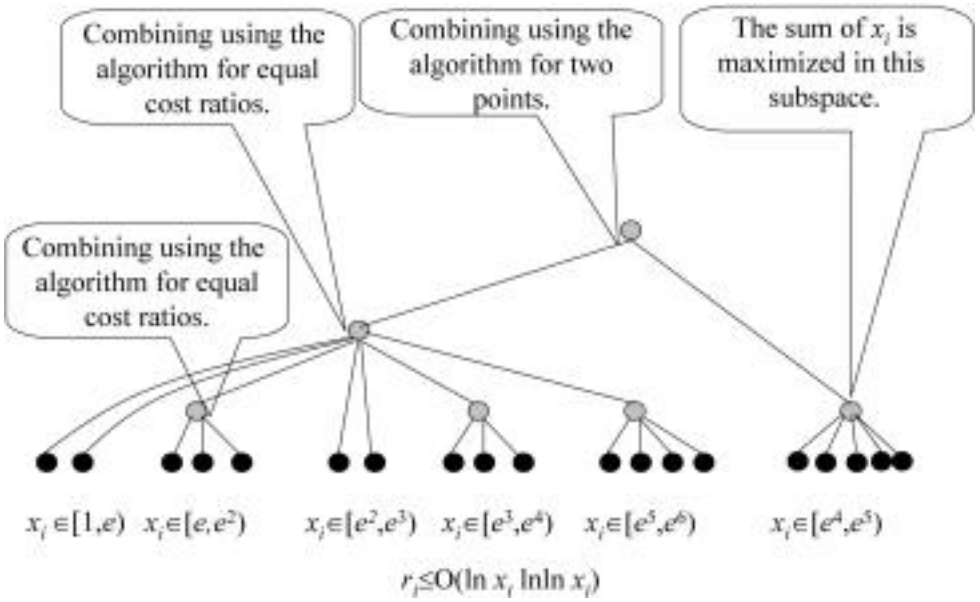


FIG. 4.1. *Schematic description of* COMBINED.

If $b' = 1$ we choose COMBINED to be $A(S_1)$, and we are done. If $b' \geq 2$, let $\tilde{M} = \cup_{i=2}^{b'} S_i$. We want to construct an algorithm, $A(\tilde{M})$, for $U(\tilde{M})$. If $b' = 2$, we choose $A(\tilde{M})$ to be $A(S_2)$. Otherwise, we apply Theorem 3.1 on $\tilde{M}$ with the partition $\{S_2, \ldots, S_{b'}\}$. We define $\hat{M}$ from Theorem 3.1 to be $\mathcal{U}_{b'-1}^d$. Likewise, $\hat{A}$ from Theorem 3.1 is the application of the (1/5)-variant of ODDEXPONENT on $\hat{U} = (\mathcal{U}_{b'-1}^d; r(S_2), \ldots, r(S_{b'}))$. Let $r(\tilde{M})$ denote the competitive ratio of $\hat{A}$.

Next, we choose the partition $\{S_1, \tilde{M}\}$ of $\mathcal{U}_b^d$. We combine the two algorithms $A(S_1)$ and $A(\tilde{M})$ using the (1/10)-variant of TWOSTABLE (this is the $\hat{A}$ required in Theorem 3.1) on the UMTS $(\mathcal{U}_2^d; r(S_1), r(\tilde{M}))$ (the UMTS $\hat{U}$ of Theorem 3.1). We denote the competitive ratio of $\hat{A}$ by r. The resulting combined algorithm, $A(M)$, is our final algorithm, COMBINED.

LEMMA 4.4. *Given that* $x = \sum_i x_i$, $r_i \leq 100s \ln x_i \ln\ln x_i$, *and* $x_i \geq e^{e^6+1}$, *algorithm* COMBINED *for the UMTS* $U = (\mathcal{U}_b^d; r_1, \ldots, r_b; s)$ *is* $(1, 1/2)$*-constrained and* r*-competitive, where* $r \leq 100s \ln x \ln\ln x$.

Proof. As before, without loss of generality, we assume $s = 1$. First we calculate the constraints of the algorithm.

From Lemma 4.1 and Lemma 3.10, $A(S_i)$ is $(1/10, 1/10)$-constrained for every $1 \leq i \leq b'$. We would like to show that $A(\tilde{M})$ is $(1/2, 3/10)$-constrained. If $b' = 2$, then it is obviously $(1/10, 1/10)$-constrained. Otherwise, $(b' > 2)$, the combining algorithm for $\tilde{M}$, is the (1/5)-variant of ODDEXPONENT, which is $(1/5, 1/5)$-constrained. Hence, from (3.3), $\beta \leq 1/5+1/10+1/10+1/10 = 1/2$, and from (3.4), $\eta \leq 1/5+1/10 = 3/10$. From Corollary 3.2, $A(\tilde{M})$ is $r(\tilde{M})$-competitive.

The (β, η)-constraints of algorithm COMBINED are calculated as follows: The (1/10)-variant of TWOSTABLE is $(1/10, 2/10)$-constrained; therefore $\beta = 1/10+1/10+1/2+3/10 = 1$ and $\eta = 2/10+3/10 = 1/2$. From Corollary 3.2, $A(M)$ is r-competitive.

To summarize, COMBINED is an $(1, 1/2)$-constrained and r-competitive algorithm for the UMTS U.

It remains to prove the bound on r. First we show that for all $1 \le j \le b'$, $r(S_j) \le 100s \ln x(S_j) \ln\ln x(S_j)$. If $|S_j| = 1$, we are done. Otherwise, $|S_j| \ge \ln x$, and $S_j = Q_\ell$ for some ℓ.

$$
\begin{aligned}
r(S_j) &\le 100 \ln e^\ell \ln\ln e^\ell + 6 \cdot 10 \ln |S_j| && (4.1)\\
&\le 100\big(\ln e^{\ell-1} \ln\ln e^{\ell-1} + \ln \ell + \tfrac{1}{\ell-1} \ln e^{\ell-1}\big) + 60 \ln |S_j| \\
&\le 100\big(\ln e^{\ell-1} \ln\ln e^{\ell-1} + \ln\ln x + \tfrac{60}{100} \ln |S_j| + 1\big) && (4.2)\\
&\le 100\big(\ln e^{\ell-1} \ln\ln e^{\ell-1} + 2 \ln |S_j|\big) && (4.3)\\
&\le 100 \ln(|S_j| e^{\ell-1}) \ln\ln(|S_j| e^{\ell-1}) \\
&\le 100 \ln x(S_j) \ln\ln x(S_j). && (4.4)
\end{aligned}
$$

Inequality (4.1) is derived as follows. Since $S_j = Q_\ell$, it follows that $r_i \le 100s \ln e^\ell \ln\ln e^\ell$ for all $v_i \in S_j$. By the bound on the competitive ratio of the (1/10) variant of OddExponent (See Lemma 4.1 and Lemma 3.10) we obtain (4.1). Inequality (4.2) follows since $\ell \le \ln x$. Inequality (4.3) follows because $\ln |S_j| \ge \ln\ln x$, and $\ln\ln x \ge 6$. The last inequality follows because $e^{\ell-1}$ is a lower bound on x_i for $v_i \in S_j$, and thus $|S_j| e^{\ell-1} \le x(S_j)$.

Observe that $b' \le \ln^2 x$, as there are at most $\ln x$ sets Q_i, and each such set contributes at most $\ln x$ sets S_i to P. We next derive a bound on $r(\tilde{M})$.

$$
\begin{aligned}
r(\tilde{M}) &\le \max_{2 \le i \le b'} r(S_i) + 6 \cdot 5 \cdot \ln(b' - 1) && (4.5)\\
&\le 100 \cdot \ln x(S_2) \ln\ln x + 30 \cdot (2 \ln\ln x) && (4.6)\\
&= 100(\ln x(S_2) + 0.6) \ln\ln x.
\end{aligned}
$$

Inequality (4.5) follows since the algorithm used is a (1/5)-variant of OddExponent. Inequality (4.6) follows by using the previously derived bound on $r(S_i)$ and noting that $x(S_2)$ is maximal amongst $x(S_2), \ldots, x(S_{b'})$ and that $x(S_i) \le x$.

From Lemma 3.10 we know that the competitive ratio of the (1/10)-variant of TwoStable is $f(10, r(S_1), r(\tilde{M}))$, where f is the function as given in Proposition 4.3. We give an upper bound on $f(10, r(S_1), r(\tilde{M}))$ using Proposition 4.3. To do this we need to find values y_1 and y_2 such that

$$
\begin{aligned}
r(S_1) &\le 100 \ln x(S_1) \ln\ln x = 2 \cdot 10(\ln y_1 + 1), \\
r(\tilde{M}) &\le 100(\ln x(\tilde{M}) + 0.6) \ln\ln x = 2 \cdot 10(\ln y_2 + 1).
\end{aligned}
$$

Indeed, the following values satisfy the conditions above: $y_1 = x(S_1)^{5 \ln\ln x}/e$ and $y_2 = (e^{0.6} x(\tilde{M}))^{5 \ln\ln x}/e$. Using Proposition 4.3 we get a bound on r as follows:

$$
\begin{aligned}
r &\le 2 \cdot 10(\ln(y_1 + y_2) + 1) && (4.7)\\
&\le 20 \ln\big(x(S_1)^{5 \ln\ln x} + (e^{0.6} x(\tilde{M}))^{5 \ln\ln x}\big) \\
&\le 20 \ln\big(x(S_1)^{5 \ln\ln x} + (2^{5 \ln\ln x} - 1) x(\tilde{M})^{5 \ln\ln x}\big) && (4.8)\\
&\le 20 \ln\big((x(S_1) + x(\tilde{M}))^{5 \ln\ln x}\big) && (4.9)\\
&\le 100 \ln x \ln\ln x.
\end{aligned}
$$

Inequality (4.7) follows from Proposition 4.3. Inequality (4.8) follows because $\ln\ln x \ge 6$. Inequality (4.9) follows since, in general, for $a \ge b > 0$ and $z \ge 1$, $a^z + (2^z - 1) b^z \le (a + b)^z$. This is because for $a = b$ it is an equality, and the

derivative with respect to a of the right-hand side is clearly larger than the derivative with respect to a of the left-hand side. ∎

Next, we present a better algorithm when all the cost ratios but one are equal.

LEMMA 4.5. *Given a UMTS $U = (\mathcal{U}_b^d; r_1, r_2, \ldots, r_b)$ with $r_2 = r_3 = \cdots = r_b$, there exists a $(1, 3/5)$-constrained and r-competitive online algorithm,* WCOMBINED, *where*

$$r = 30\Big(\ln\big(e^{\frac{r_1}{30}-\frac{1}{3}} + (b-1)e^{\frac{r_2}{30}-\frac{1}{3}}\big) + \tfrac{1}{3}\Big).$$

Proof. The proof is a simplified version of the proof of Lemma 4.4, and we just sketch it here. We define x_1, x_2 such that

$$r_1 = 30(\ln x_1 + \tfrac{1}{3}) = 2 \cdot 5 \cdot (\ln x_1^3 + 1), \quad r_2 = 30(\ln x_2 + \tfrac{1}{3}) = 2 \cdot 5 \cdot (\ln x_2^3 + 1).$$

Let $\tilde{M} = \{v_2, \ldots, v_b\}$. We use a (1/5)-variant of ODDEXPONENT on the UMTS $U(\tilde{M})$. The competitive ratio of this algorithm is at most

$$r(\tilde{M}) \le r_2 + 30\ln(b-1) \le 30\big(\ln((b-1)x_2) + \tfrac{1}{3}\big) = 10\big(\ln((b-1)x_2)^3 + 1\big),$$

and it is $(1/5, 1/5)$-constrained. We combine it with the trivial algorithm for $U(\{v_1\})$ using a (1/5)-variant of algorithm TWOSTABLE. The resulting algorithm is $(1, 3/5)$-constrained, and by Proposition 4.3 we have

$$r \le 10(\ln(x_1^3+((b-1)x_2)^3+1) \le 10(\ln(x_1+(b-1)x_2)^3+1) = 30(\ln(x_1+(b-1)x_2)+\tfrac{1}{3}).$$

Substituting for x_i gives the required bound. ∎

5. Applications.

5.1. An $O((\log n \log\log n)^2)$-competitive algorithm for MTSs. Bartal [1] defines a class of decomposable spaces called HSTs.[1]

DEFINITION 5.1. *For $k \ge 1$, a k-HST is a metric space defined on the leaves of a rooted tree T. Associated with each vertex $u \in T$ is a real-valued label $\Delta(u) \ge 0$, and $\Delta(u) = 0$ if and only if u is a leaf of T. The labels obey the rule that for every vertex v, a child of u, $\Delta(v) \le \Delta(u)/k$. The distance between two leaves $x, y \in T$ is defined as $\Delta(\mathrm{lca}(x, y))$, where $\mathrm{lca}(x, y)$ is the least common ancestor of x and y in T. Clearly, this is a metric.*

Bartal [1, 2] shows how to approximate any metric space using an efficiently constructible probability distribution over a set of k-HSTs . His result allows us to reduce a MTS problem on an arbitrary metric space to MTS problems on HSTs. Formally, he proves the following theorem.

THEOREM 5.2 (see [2]). *Suppose there is an r-competitive algorithm for any n-point k-HST metric space. Then there exists an $O(rk \log n \log\log n)$-competitive randomized algorithm for* any *n-point metric space.*

Thus, it is sufficient to construct an online algorithm for a MTS, where the underlying metric space is a k-HST. Following [3] we use the UMTS model to obtain an online algorithm for a MTS over a k-HST metric space.

[1] The definition given here for k-HST differs slightly from the original definition given in [1]. We choose the definition given here for simplicity of the presentation. For $k > 1$ the metric spaces given by these two definitions approximate each other to within a factor of $k/(k-1)$.

ALGORITHM RHST. We define the algorithm RHST(T) on the metric space $M(T)$, where T is a k-HST with $k \geq 5$. Algorithm RHST(T) is defined inductively on the size of the underlying HST, T.

When $|M(T)| = 1$, RHST(T) serves all task sequences optimally. It is $(0,0)$-constrained. Otherwise, let the children of the root of T be $v_1, \ldots, v_b$, and let T_i be the subtree rooted at v_i. Denote $d = \Delta(T)$, and so $\mathrm{diam}(T_i) \leq d/k$. Every algorithm RHST($T_i$) is an algorithm for the UMTS $U_i = (M(T_i); 1, \ldots, 1; 1)$.

We construct a metric space $\hat{M} = \mathcal{U}_b^d$ and define cost ratios $r_1, \ldots, r_b$, where $r_i = r(T_i)$ is the competitive ratio of RHST(T_i). We now use Theorem 3.1 to combine algorithms RHST(T_i). The role of $\hat{A}$ is played by the $(1/2)$-variant of COMBINED on the UMTS $\hat{U} = (\hat{M}; r_1, \ldots, r_b; 1)$. The combined algorithm is a RHST(T) on the UMTS $(M(T); 1, \ldots, 1; 1)$.

We remark that the application of Theorem 3.1 requires that the algorithms will be constrained. We show that this is true in the following lemma.

LEMMA 5.3. *The algorithm* RHST(T) *is* $O(\ln n \ln\ln n)$, *where* $n = |M(T)|$.

Proof. Let $n' = e^{e^6+1} n$. We prove by induction on the depth of the tree that RHST(T) is $(1,1)$-constrained and $200 \ln n' \ln\ln n'$-competitive.

When $|M(T)| = 1$, it is obvious. Otherwise, let $n_i = |M(T_i)|$, $n_i' = e^{e^6+1} n_i$, and $n' = \sum_i n_i'$. We assume inductively that each of the RHST(T_i) algorithms is $(1,1)$-constrained and $200 \ln n_i' \ln\ln n_i'$-competitive on $M(T_i)$. The combined algorithm, RHST(T), is (β, η)-constrained. From (3.3), and given that $k \geq 5$, we get that

$$\beta \leq \max\{1, \tfrac{1}{2} + \tfrac{1}{k} + \tfrac{1}{k} + \tfrac{1}{2k}\} \leq \max\{1, 1\} = 1.$$

From (3.4) we obtain that $\eta \leq \frac{1}{2} + \frac{1}{k} \leq 1$ for $k \geq 5$. This proves that the algorithm is well defined and $(1,1)$-constrained.

We next bound the competitive ratio using Lemma 4.4. Lemma 3.10 implies that the competitive ratio obtained by the $(1/2)$-variant of COMBINED on $(\hat{M}; r_1, \ldots, r_b)$ is the same as the competitive ratio attained by COMBINED on $(\hat{M}; r_1, \ldots, r_b; 2)$. The values $(x_i)_i$ computed by COMBINED are at most $(n_i')_i$, respectively. Hence it follows from Lemma 4.4 that the competitive ratio of RHST(T) is at most $100 \cdot 2 \ln x \ln\ln x \leq 200 \ln n' \ln\ln n'$, since $x = \sum_i x_i$. ∎

Since every HST T can be 5-approximated by a 5-HST T' (see [2]), the bound we have just proved holds for any HST.

Combining Theorem 5.2 with Lemma 5.3, we get the following theorem.

THEOREM 5.4. *For any MTS over an n-point metric space, the randomized competitive ratio is* $O((\log n \log\log n)^2)$.

5.2. K-weighted caching on $K+1$ points. Weighted caching is a generalized paging problem, where there is a different cost to fetch different pages. This problem is equivalent to the K-server problem on a *star metric space* [23, 9]. A star metric space is derived from a depth-one tree with distances on the edges, the points of the metric space are the leaves of the tree, and the distance between a pair of points is the length of the (2 edge) path between them. This is so, since we can assign any edge (r, u) in the tree a weight of half the fetch cost of u. Together, an entrance of a server into a leaf from the star's middle-point (page in) and leaving the leaf to the star's middle-point (page out) have the same cost of fetching the page.

The K-server problem on a metric space of $K+1$ points is a special case of the MTS problem on the same metric space, and hence any upper bound for the MTS translates to an upper bound for the corresponding K-server problem.

Given a star metric space M, we 12-approximate it with a 6-HST T. T has the special structure that for every internal vertex, all children, except perhaps one, are leaves. It is not hard to see that one can find such a tree T such that for any $u, v \in M$, $\mathrm{d}_M(u,v) \le \mathrm{d}_T(u,v) \le 12 \cdot \mathrm{d}_M(u,v)$. Essentially, the vertices furthest away from the root (up to a factor of 6) in the star are children of the root of T, and the last child of the root is a recursive construction for the rest of the points.

We now follow the construction of RHST given in the previous section, on a 6-HST T, except that we make use of (1/2)-variant of WCOMBINED rather than (1/2)-variant of COMBINED. The special structure of T implies that all the children of an inner vertex, except perhaps one, are leaves and therefore have a trivial 1-competitive algorithm on their "subspaces." Hence we can apply WCOMBINED. Using Lemma 4.5 with induction on the depth of the tree, it is easy to bound from above the competitive ratio on a 6-HST with $K+1$ leaves by $60(\ln(K+1)+1/3)$.

Combining the above with the lower bound of [9] we obtain the following theorem.

THEOREM 5.5. *The competitive ratio for the K-weighted caching problem on $K+1$ points is $\Theta(\log K)$.*

5.3. A MTS on equally spaced points on the line. The metric space of n equally spaced points on the line is considered important because of its simplicity and the practical significance of the k-server on the line (for which this problem is a special case). The best lower bound currently known on the competitive ratio is $\Omega(\log n/\log\log n)$ [10]. Previously, the best upper bound known was $O(\log^3 n/\log\log n)$ due to [3].

We are able to slightly improve the upper bound on the competitive ratio from section 5.1 to $O(\log^2 n)$. Bartal [1] proves that n equally spaced points on the line can be $O(\log n)$ probabilistically embedded into a set of *binary* 4-HSTs. We present an $O(\log n)$-competitive randomized algorithm for binary 4-HST, similar to RHST except that we make use of (1/4)-variant of TWOSTABLE instead of (1/2)-variant of COMBINED. Similar arguments show that this algorithm is $(1,1)$-constrained, and using Proposition 4.3 we conclude that the algorithm is $8\ln n$-competitive. Combining the probabilistic embedding into binary 4-HST with the algorithm for binary 4-HST we obtain the following theorem.

THEOREM 5.6. *The competitive ratio of the MTS problem on metric space of n equally spaced points on the line is $O(\log^2 n)$.*

6. Concluding remarks. This paper presents algorithms for the MTS problem and related problems with significantly improved competitive ratios. An obvious avenue of research is to further improve the upper bound on the competitive ratio for the MTS problem. A slight improvement to the competitive ratio of the algorithm for arbitrary n-point metric spaces is reported in [6]. The resulting competitive ratio there is $O(\log^2 n \log\log n \log\log\log n)$, and the improvement is achieved by refining the reduction from arbitrary metric spaces to HST spaces (i.e., that improvement is orthogonal to the improvement presented in this paper). However, in order to break the $O(\log^2 n)$ bound, it seems that one needs to deviate from the black box usage of Theorem 5.2. Maybe the easiest special case to start with is the metric space of equally spaced points on the line.

Another interesting line of research would be an attempt to apply the techniques of this and previous papers to the randomized k-server problem, or even for a special

case such as the randomized weighted caching on k pages problem; see also [8, 21].

Acknowledgments. We would like to thank Yair Bartal, Avrim Blum, and Steve Seiden for helpful discussions.

Note added in proof. Theorem 5.2 has been improved in [14]. Therefore the bound in Theorem 5.4 is also improved to $O(\log^2 n \log\log n)$.

REFERENCES

[1] Y. Bartal, *Probabilistic approximation of metric space and its algorithmic application*, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, Burlington, VT, 1996, pp. 183–193.

[2] Y. Bartal, *On approximating arbitrary metrics by tree metrics*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 161–168.

[3] Y. Bartal, A. Blum, C. Burch, and A. Tomkins, *A polylog(n)-competitive algorithm for metrical task systems*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 711–719.

[4] Y. Bartal, B. Bollobás, and M. Mendel, *A Ramsey-type theorem for metric spaces and its application for metrical task systems and related problems*, in Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, Las Vegas, NV, 2001, pp. 396–405.

[5] Y. Bartal, N. Linial, M. Mendel, and A. Naor, *On metric Ramsey-type phenomena*, in Proceedings of the 35th Annual ACM Symposium on the Theory of Computing, San Diego, CA, 2003, pp. 463–472.

[6] Y. Bartal and M. Mendel, *Multi-embeddings and path-approximation of metric spaces*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, pp. 424–433.

[7] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson, *On the power of randomization in on-line algorithms*, Algorithmica, 11 (1994), pp. 2–14.

[8] A. Blum, C. Burch, and A. Kalai, *Finely-competitive paging*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, NY, 1999, pp. 450–458.

[9] A. Blum, M. L. Furst, and A. Tomkins, *What to Do with Your Free Time: Algorithms for Infrequent Requests and Randomized Weighted Caching*, manuscript, 1996.

[10] A. Blum, H. Karloff, Y. Rabani, and M. Saks, *A decomposition theorem for task systems and bounds for randomized server problems*, SIAM J. Comput., 30 (2000), pp. 1624–1661.

[11] A. Borodin, N. Linial, and M. Saks, *An optimal online algorithm for metrical task systems*, J. Assoc. Comput. Mach., 39 (1992), pp. 745–763.

[12] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan, *New results on server problems*, SIAM Discrete Math., 4 (1991), pp. 172–181.

[13] M. Chrobak and L. L. Larmore, *The server problem and on-line games*, in On-line Algorithms, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 7, L. A. McGeoch and D. D. Sleator, eds., AMS, Providence, RI, 1991, pp. 11–64.

[14] J. Fakcharoenphol, S. Rao, and K. Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, CA, 2003, pp. 448–455.

[15] S. Irani, *Randomized weighted caching with two page weights*, Algorithmica, 32 (2002), pp. 624–640.

[16] S. Irani and S. Seiden, *Randomized algorithms for metrical task systems*, Theoret. Comput. Sci., 194 (1998), pp. 163–182.

[17] A. Karlin, M. Manasse, L. Rudolph, and D. D. Sleator, *Competitive snoopy caching*, Algorithmica, 3 (1988), pp. 79–119.

[18] H. Karloff, Y. Rabani, and Y. Ravid, *Lower bounds for randomized k-server and motion-planning algorithms*, SIAM J. Comput., 23 (1994), pp. 293–312.

[19] M. Manasse, L. A. McGeoch, and D. Sleator, *Competitive algorithms for server problems*, J. Algorithms, 11 (1990), pp. 208–230.

[20] S. Seiden, *Unfair problems and randomized algorithms for metrical task systems*, Inform. and Comput., 148 (1999), pp. 219–240.

[21] S. Seiden, *A general decomposition theorem for the k-server problem*, in Proceedings of the 9th Annual European Symposium on Algorithms, Aarhus, Denmark, 2001, Lecture Notes in Comput. Sci. 2161, Springer, Berlin, 2001, pp. 86–97.

[22] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Commun. ACM, 28 (1985), pp. 202–208.

[23] N. E. Young, *The k-server dual and loose competitiveness for paging*, Algorithmica, 11 (1994), pp. 525–541.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1423–1435

MATRIX ROUNDING UNDER THE L_p-DISCREPANCY MEASURE AND ITS APPLICATION TO DIGITAL HALFTONING*

TETSUO ASANO†, NAOKI KATOH‡, KOJI OBOKATA†, AND TAKESHI TOKUYAMA§

Abstract. We study the problem of rounding a real-valued matrix into an integer-valued matrix to minimize an L_p-discrepancy measure between them. To define the L_p-discrepancy measure, we introduce a family $\mathcal{F}$ of regions (rigid submatrices) of the matrix and consider a hypergraph defined by the family. The difficulty of the problem depends on the choice of the region family $\mathcal{F}$. We first investigate the rounding problem by using integer programming problems with convex piecewise-linear objective functions and give some nontrivial upper bounds for the L_p discrepancy. We propose "laminar family" for constructing a practical and well-solvable class of $\mathcal{F}$. Indeed, we show that the problem is solvable in polynomial time if $\mathcal{F}$ is the union of two laminar families. Finally, we show that the matrix rounding using L_1 discrepancy for the union of two laminar families is suitable for developing a high-quality digital-halftoning software.

Key words. approximation algorithm, digital halftoning, discrepancy, linear programming, matrix rounding, network flow, totally unimodular

AMS subject classifications. 68R05, 68W25, 68W40, 90C05, 90C27

DOI. 10.1137/S0097539702417511

1. Introduction.

Rounding is an important operation in numerical computation and plays key roles in digitization of analog data. Rounding of a real number a is basically a simple problem: We round it to either $\lfloor a \rfloor$ or $\lceil a \rceil$, and we usually choose the one nearer to a. However, we often encounter a datum consisting of more than one real number instead of a singleton. If it has n numbers, we have 2^n choices for rounding since each number is rounded into either its floor or ceiling. If the original data set has some feature, we need to choose a rounding so that the rounded result inherits as much of the feature as possible. The feature is described by using some combinatorial structure; we indeed consider a hypergraph $\mathcal{H}$ on the set. A typical input set is a multidimensional array of real numbers, and we consider a hypergraph whose hyperedges are its subarrays with contiguous indices. In this paper, we focus on two-dimensional arrays; in other words, we consider rounding problems on matrices.

1.1. Rounding problem and discrepancy measure.

Given an $M \times N$ matrix $A = (a_{ij})_{1 \le i \le M, 1 \le j \le N}$ of real numbers, its rounding is a matrix $B = (b_{ij})_{1 \le i \le M, 1 \le j \le N}$ of integral values such that b_{ij} is either $\lfloor a_{ij} \rfloor$ or $\lceil a_{ij} \rceil$ for each (i, j). There are 2^{MN} possible roundings of a given A, and we would like to find an optimal rounding with respect to a given criterion. This is called the *matrix rounding problem*. Without loss of generality, we can assume that each entry of A is in the closed interval $[0, 1]$ and each entry is rounded to either 0 or 1.

*Received by the editors November 9, 2002; accepted for publication (in revised form) June 13, 2003; published electronically September 9, 2003. An early extended abstract of this paper appeared in *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, CA, 2002. This work was supported by Grant in Aid for Scientific Research of the Ministry of Education, Culture, Sports, Science and Technology of Japan.

http://www.siam.org/journals/sicomp/32-6/41751.html

†School of Information Science, Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Tatsunokuchi, 923-1292 Japan (t-asano@jaist.ac.jp, obokata@jaist.ac.jp).

‡Graduate School of Engineering, Kyoto University, Yoshida-Honmachi, Sakyoku, Kyoto, 606-8501 Japan (naoki@archi.kyoto-u.ac.jp).

§Graduate School of Information Sciences, Tohoku University, Aramaki, Aobaku, Sendai, 980-8579 Japan (tokuyama@dais.is.tohoku.ac.jp).

In order to give a criterion to determine the quality of roundings, we define a distance in the space of all $[0,1]$-valued $M \times N$ matrices. Let $n = MN$. Let $\mathcal{F}$ be a family of *regions* (i.e., subsets) of the $M \times N$ integer grid G_{MN}. Let $\mathcal{A} = \mathcal{A}(G_{MN})$ be the space of all $[0,1]$-valued matrices with the index set G_{MN}, and let $\mathcal{B} = \mathcal{B}(G_{MN})$ be its subset consisting of all $\{0,1\}$-valued matrices. Let R be a region in $\mathcal{F}$.[1] For an element $A \in \mathcal{A}$, let $A(R)$ be the sum of entries of A located in the region R, that is,

$$A(R) = \sum_{(i,j)\in R} a_{ij}.$$

We define a distance $Dist_p^{\mathcal{F}}(A, A')$ between two elements A and A' in $\mathcal{A}$ for a positive integer p by

$$Dist_p^{\mathcal{F}}(A, A') = \left[\sum_{R\in\mathcal{F}} |A(R) - A'(R)|^p\right]^{1/p}.$$

The distance is called the L_p distance with respect to $\mathcal{F}$. The L_∞ distance with respect to $\mathcal{F}$ is defined by

$$Dist_\infty^{\mathcal{F}}(A, A') = \lim_{p\to\infty} Dist_p^{\mathcal{F}}(A, A') = \max_{R\in\mathcal{F}} |A(R) - A'(R)|.$$

Using the notations above, we can formally define the matrix rounding problem.

L_p-optimal matrix rounding problem. $\mathcal{P}(G_{MN}, \mathcal{F}, p)$: Given a $[0,1]$-matrix $A \in \mathcal{A}$, a family $\mathcal{F}$ of subsets of G_{MN}, and a positive integer p, find a $\{0,1\}$-matrix $B \in \mathcal{B}$ that minimizes

$$Dist_p^{\mathcal{F}}(A, B) = \left[\sum_{R\in\mathcal{F}} |A(R) - B(R)|^p\right]^{1/p}.$$

Also, we are interested in the following combinatorial problem.

L_p-discrepancy bound. Given a $[0,1]$-matrix $A \in \mathcal{A}$, a family $\mathcal{F}$ of subsets of G_{MN}, and a positive integer p, investigate upper and lower bounds of

$$\mathcal{D}(G_{MN}, \mathcal{F}, p) = \sup_{A\in\mathcal{A}} \min_{B\in\mathcal{B}} Dist_p^{\mathcal{F}}(A, B).$$

The pair $(G_{MN}, \mathcal{F})$ defines a hypergraph on G_{MN}, and $\mathcal{D}(G_{MN}, \mathcal{F}, \infty)$ is called the *inhomogeneous discrepancy* of the hypergraph [6]. Abusing the notation, we call $\mathcal{D}(G_{MN}, \mathcal{F}, p)$ the (inhomogeneous) L_p discrepancy of the hypergraph and also often call $Dist_p^{\mathcal{F}}(A, B)$ the L_p-discrepancy measure of (the quality of) the output B with respect to $\mathcal{F}$.

1.2. Motivation and our application. The most popular example of the family $\mathcal{F}$ is the set of all rectangular subregions in G_{MN} (i.e., the set of all rigid submatrices), and the corresponding L_∞-discrepancy measure is utilized in many application areas such as Monte Carlo simulation and computational geometry. Unfortunately, if we consider the family of all rectangular subregions, the discrepancy bound (for the

[1] Strictly speaking, R can be any subset of G_{MN}. Although we implicitly assume that R forms some connected portion on the grid G_{MN}, the connectivity assumption is not used throughout the paper.

L_∞ measure) is known to be $\Omega(\log n)$ and $O(\log^3 n)$. See Beck and Sós's survey [6] for the theory. It seems hard to find an optimal solution to minimize the discrepancy. In fact, it is NP-hard [2].

Therefore, we seek a family of regions for which low discrepancy rounding is useful in an important application and also can be computed in polynomial time. For the application, L_∞ rounding is not always suitable, and L_p discrepancy (with $p = 1$ or 2) is preferable. For the purpose, we present a geometric structure of a family of regions reflecting the combinatorial discrepancy bound and computational difficulty of the matrix rounding problem.

In particular, we focus on the digital-halftoning application of the matrix rounding problem, where we should consider smaller families of rectangular subregions as $\mathcal{F}$. More precisely, the input matrix represents a digital (gray) image, where a_{ij} represents the brightness level of the (i, j)-pixel in the $M \times N$ pixel grid. Typically, M and N are between 256 and 4096, and a_{ij} is an integral multiple of $1/256$: This means that we use 256 brightness levels. If we want to send an image using fax or print it out by a dot (or ink-jet) printer, brightness levels available are limited. Instead, we replace A by an integral matrix B so that each pixel uses only two brightness levels. Here, it is important that B looks similar to A; in other words, B should be a good approximation of A.

For each pixel (i, j), if the average brightness level of B in each of its neighborhoods (regions containing (i, j) in a suitable family of regions) is similar to that of A, we can expect that B is a good approximation of A. For this purpose, the set of all rectangles is not suitable (i.e., it is too large), and we may use a more compact family. Moreover, since human vision detects global features, the L_1 or L_2 measure should be better than the L_∞ measure to obtain a clear output image. This intuition is supported by our experimental results; for example, edges of objects are often blurred in the output based on the L_∞-discrepancy measure, while they are sharply displayed if we use the L_1-discrepancy measure.

1.3. Known results on L_∞ measure. For the L_∞ measure, the following beautiful combinatorial result is classically known.

THEOREM 1.1 (Baranyai [5]). *Given a real-valued matrix $A = (a_{ij})$ and a family $\mathcal{F}$ of regions consisting of all rows, all columns, and the whole matrix, there exists an integer-valued matrix $B = (b_{ij})$ such that $|A(R) - B(R)| < 1$ holds for every $R \in \mathcal{F}$.*

Translating the theorem in our terminologies, the L_∞ discrepancy of the matrix rounding problem for the family of regions consisting of all rows, all columns, and the whole matrix is bounded by 1. Also, the combinatorial structure and algorithmic aspects of roundings of (one-dimensional) sequences with respect to the L_∞-discrepancy measure are investigated in recent studies [2, 19].

The *incidence matrix* $\mathcal{C}(G_{MN}, \mathcal{F}) = (\mathcal{C}_{ij})$ of the hypergraph $(G_{MN}, \mathcal{F})$ is defined by $\mathcal{C}_{ij} = 1$ if the jth element of G_{MN} belongs to the ith region R_i in $\mathcal{F}$ and 0 otherwise.[2] A hypergraph is called *unimodular* if its incidence matrix is totally unimodular, where a matrix $\mathcal{C}$ is *totally unimodular* if the determinant of each square submatrix of $\mathcal{C}$ is equal to 0, 1, or -1.

Both the Baranyai problem and the sequence rounding problems correspond to rounding problems with respect to unimodular hypergraphs. The L_∞-discrepancy problem can be formulated as an integer programming problem, and the unimodularity implies that its relaxation has an integral solution. A classical theorem of

[2] We implicitly assume a one-dimensional ordering of elements in G_{MN}.

Ghouila-Houri [10] implies that unimodularity is a necessary and sufficient condition for the existence of a rounding with a L_∞ discrepancy less than 1. Moreover, the following sharpened result is given by Doerr [7].

THEOREM 1.2. *If $(G_{MN}, \mathcal{F})$ is a unimodular hypergraph, there exists a rounding $B = (b_{ij})$ of $A = (a_{ij})$ satisfying*

$$|A(R) - B(R)| < \min\left\{1 - \frac{1}{n+1}, 1 - \frac{1}{m}\right\}$$

for every $R \in \mathcal{F}$, where $n = MN$, $m = |\mathcal{F}|$.

This bound is sharp. Moreover, L_∞-optimal rounding can be computed in polynomial time if $\mathcal{F}$ is unimodular.

1.4. Our results. We would like to consider the L_p-discrepancy measure instead of the L_∞-discrepancy measure. If the hypergraph is unimodular, an $|\mathcal{F}|^{1/p}$ upper bound for the L_p discrepancy can be derived from Theorem 1.2 trivially. We first improve the upper bound to $\frac{1}{2}|\mathcal{F}|^{1/p}$ for $p \leq 3$ and show that the bound is tight. We also consider the family $\mathcal{F}$, consisting of all 2×2 rigid submatrices, for which the matrix rounding problem is known to be NP-hard [2] (accordingly, the family is not unimodular).

Next, we consider the optimization problem. If the hypergraph is unimodular, the rounding minimizing the L_p discrepancy can be computed in polynomial time by translating it to a separable convex programming problem and applying known general algorithms [11, 12]. However, we want to define a class of region families for which we can compute the optimal solution more efficiently, as well as a class that is useful in applications (in particular, the digital-halftoning application). We consider the union of two laminar families (defined in section 3) and show that the matrix rounding problem can be formulated into a minimum cost flow problem, and hence solved in polynomial time. Finally, we implemented the algorithm using LEDA [14]. Some output pictures of the algorithm applying to the digital-halftoning problem are included.

2. Mathematical programming formulations.

2.1. Formulation as a piecewise-linear separable convex programming problem. We give a formulation of the L_p-discrepancy problem into an integer convex programming problem where the objective function is a separable convex function, i.e., a sum of univariate convex functions.

Introducing a new variable $y_i = B(R_i) = \sum_{(j,k)\in R_i} b_{jk}$ for each $R_i \in \mathcal{F}$, the problem $\mathcal{P}(G_{MN}, \mathcal{F}, p)$ is described in the following form:

$$\begin{aligned} \text{(P1)} : \text{minimize} \quad & \left[\sum_{R_i \in \mathcal{F}} |y_i - A(R_i)|^p\right]^{1/p} \\ \text{subject to} \quad & y_i = \sum_{(j,k)\in R_i} b_{jk}, \ i = 1, \ldots, m = |\mathcal{F}|, \\ \text{and} \quad & B \in \mathcal{B}(G_{MN}). \end{aligned}$$

When $p < \infty$, the objective function can be replaced with $\sum_{R_i \in \mathcal{F}} |y_i - c_i|^p$, where $c_i = A(R_i) = \sum_{(j,k)\in R_i} a_{jk}$ is a constant depending only on input values. Now $|y_i - c_i|^p$ is a convex function independent of other y_j's. The constraints $y_i = \sum_{(j,k)\in R_i} b_{jk}$,

$i = 1, \ldots, m$, are represented by $(-I, \mathcal{C}(G_{MN}, \mathcal{F}))Y = 0$ using the incidence matrix $\mathcal{C}(G_{MN}, \mathcal{F})$ defined in section 1.3, where $Y = (y_1, \ldots, y_m, b_{11}, \ldots, b_{MN})^T$ and I is an identity matrix.

Although the objective function is now a separable convex function, its nonlinearity makes it difficult to analyze the properties of the solution. Thus, we apply the idea of Hochbaum and Shanthikumar [11] to replace $|y_i - c_i|^p$ with a piecewise-linear convex continuous function $f_i(y_i)$ which is equal to $|y_i - c_i|^p$ for each integral value of y_i in $[0, |R_i|]$. This is because we need only integral solutions, and, if each b_{p_j} is integral, y_i must be a nonnegative integer less than or equal to $|R_i|$. Typically for $p = 1$, $f_i(y_i)$ is illustrated in Figure 1.

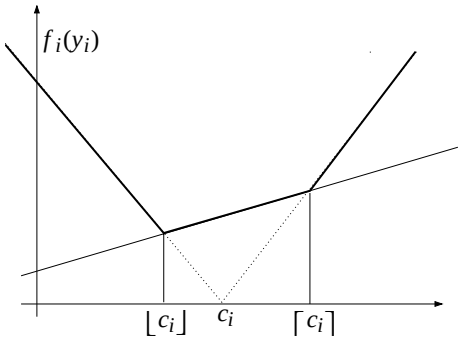


FIG. 1. *Conversion of the convex objective function $|y_i - c_i|$ into a piecewise-linear convex function $f_i(y_i)$ with integral breakpoints (shown in bold lines).*

Thus, we obtain the following problem (P2):

$$\begin{aligned} \text{(P2)} : \text{minimize} \quad & \sum_{R_i \in \mathcal{F}} f_i(y_i) \\ \text{subject to} \quad & y_i = \sum_{(j,k) \in R_i} b_{jk}, \ i = 1, \ldots, m = |\mathcal{F}|, \\ \text{and} \quad & B \in \mathcal{B}(G_{MN}). \end{aligned}$$

Thus, we can formulate the problem into an integer programming problem where the objective function is a separable piecewise-linear convex function.

2.2. Relaxation and totally unimodularity. Let (P3) be the continuous relaxation obtained from (P2) by replacing the integral condition of b_{ij} with the condition $0 \le b_{ij} \le 1$. Note that this is different from the continuous relaxation of (P1), since the objective function of (P2) is larger than that of (P1) at nonintegral values.

If the matrix is totally unimodular, (P3) has an integral solution by the theorem below. This is a key to derive discrepancy bounds and also algorithms.

THEOREM 2.1 (Hochbaum and Shanthikumar [11]). *A nonlinear separable convex optimization problem* $\min\{\sum_{i=1}^{n} f_i(x_i) \mid Ax \ge b\}$ *on linear constraints with a totally unimodular matrix A can be solved in polynomial time.*

This theorem is translated into our terminologies as follows.

COROLLARY 2.2. *The matrix rounding problem $\mathcal{P}(G_{MN}, \mathcal{F}, p)$ for $p < \infty$ is solved in polynomial time in $n = MN$ if its associated incidence matrix $\mathcal{C}(G_{MN}, \mathcal{F})$ is totally unimodular.*

3. Geometric families of regions defining unimodular hypergraphs. In this section we consider interesting classes of families whose associated incidence matrices are totally unimodular. We call such a family a *unimodular family*, since the associated hypergraph is unimodular. A family $\mathcal{F} = \{R_1, R_2, \ldots, R_m\}$ is a *partition family* (or a partition) of G_{MN} if $\bigcup_{i=1}^{m} R_i = G_{MN}$ and $R_i \cap R_j = \emptyset$ for any $R_i \neq R_j$ in $\mathcal{F}$. A *k-partition family* is a family of regions on a matrix which is the union of k different partitions of G_{MN}.

A family $\mathcal{F}$ of regions on a grid G_{MN} is a *laminar family* if one of the following holds for any pair R_i and R_j in $\mathcal{F}$: (1) $R_i \cap R_j = \emptyset$, (2) $R_i \subset R_j$, or (3) $R_j \subset R_i$. The family is also called a laminar decomposition of the grid G_{MN}. In general, a *k-laminar family* is a family of regions on a matrix which is the union of k different laminar families.

PROPOSITION 3.1. *A 2-laminar family is unimodular.*

Direct applications of Proposition 3.1 lead to various unimodular families of regions. The family of regions defined in Baranyai's theorem is a 2-laminar family. Also, take any 2-partition family consisting of 2×2 regions on a matrix. For example, take all 2×2 regions with their upper left corners located in even points (where the sums of their row and column indices are even). The set of all those regions defines 2-partition families $\mathcal{F}_{even}$ and $\mathcal{F}_{odd}$, where $\mathcal{F}_{even}$ (resp., $\mathcal{F}_{odd}$) consists of all 2×2 squares with their upper left corners lying at even (resp., odd) rows (see Figure 2). This kind of family plays an important role in section 5.2 and also in our experiment.

A 3-partition family is not unimodular in general. However, there are some families which are not 2-laminar but unimodular: For example, the set of all rectangular rigid submatrices of size 2 (i.e., domino tiles) is a 4-partition family, but it is unimodular.

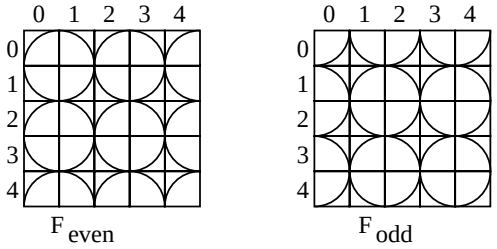


FIG. 2. *2-partition family of* 2×2 *regions.*

4. Algorithms for computing the optimal rounding. The arguments so far guarantee the polynomial-time solvability of our problem. However, we needed a more practical algorithm for our experiments that runs fast for large-scale problem instances. In this section we will show how to solve the matrix rounding problem for a 2-laminar family based on the minimum-cost flow algorithm. To improve the readability, we mainly discuss the case for the L_1-discrepancy measure.

Our main result is the following.

THEOREM 4.1. *Given a* $[0,1]$*-matrix* A *and a 2-laminar family* $\mathcal{F}$*, an optimal binary matrix* B *that minimizes the distance* $Dist_1^{\mathcal{F}}(A,B)$ *is computed in* $O(n^2 \log^2 n)$ *time, where* n *is the number of matrix elements.*

Proof. We can transform the problem into that of finding a minimum-cost circulation flow in the network defined as follows.

Let $\mathcal{F}$ be a 2-laminar family given as the union of two laminar families $\mathcal{F}_1 = \{R_0, R_1, \ldots, R_m\}$ and $\mathcal{F}_2 = \{R'_0, R'_1, \ldots, R'_{m'}\}$ over the grid G_{MN}, where R_0 and R'_0 are the entire region G_{MN}. The network to be constructed consists of three parts. The first part is an in-tree T_1 derived from $\mathcal{F}_1$ whose root is R_0; the second one is an out-tree T_2 from $\mathcal{F}_2$ whose root is R'_0; and the third part connects T_1 and T_2. The lattice structure implied by $\mathcal{F}_1$ naturally defines an in-tree T_1 such that the vertex set is the set of regions in $\mathcal{F}_1$, and there is a directed edge (R_i, R_j) if and only if $R_i \subseteq R_j$ and there is no other region R_k such that $R_j \subseteq R_k \subseteq R_i$. Then each region R_i, $i \geq 1$, has a unique outgoing edge, which is denoted by $e(R_i)$. We can similarly define T_2 for the laminar family $\mathcal{F}_2$, in which the edge direction is reversed in T_2; that is, each node R'_i, $i \geq 1$, has a unique incoming edge, which is denoted by $e(R'_i)$.

In addition, leaves of T_1 and T_2 are connected by edges corresponding to elements of G_{MN}. Because of the definition of $\mathcal{F}_1$ and $\mathcal{F}_2$, each element (k, l) of G_{MN} belongs to exactly one region in $\mathcal{F}_1$ which is a leaf in T_1 and to exactly one region in $\mathcal{F}_2$ which is a leaf in T_2. If (k, l) belongs to R_i and R'_j, then we have a directed edge $e(i, j)$ from R'_j to R_i. Finally, we draw an edge from R_0 to R'_0.

Now we define the capacity and cost coefficient of each edge. (The lower bound on the flow of each edge is defined to be 0.) The capacity of an edge $e(i, j)$ is determined simply as 1 because the value associated with an element of G_{MN} is to be rounded to 0 or 1.

Determining the cost coefficients of edges $e(R_i)$ and $e(R'_j)$ are not straightforward, although the cost coefficients of $e(i, j)$s and $e(R_0, R'_0)$ are defined to be 0s. This is because each term of the objective function depends on the difference between $B(R_i)$ and $A(R_i)$ or between $B(R'_j) - A(R'_j)$; that is, $|B(R_i) - A(R_i)|$ or $|B(R'_j) - A(R'_j)|$.

Recall the argument in section 2: To prove the polynomial-time solvability we have introduced a new variable $y_k = B(R_k) = \sum_{(i,j)\in R_k} b_{ij}$. $|B(R_k) - A(R_k)|$ is converted into $|y_k - c_k|$, where $c_k = A(R_k) = \sum_{(i,j)\in R_k} a_{ij}$ is a constant determined by input values. $|y_k - c_k|$ is further replaced by the piecewise-linear convex function $f_k(y_k)$ which coincides with $|y_k - c_k|$ at each integral value of y_k.

To reflect the new form of the objective function (see Figure 1), we replace each edge $e(R_k)$ by three parallel edges with different capacities and costs: $e_1(R_k)$ has capacity $\lfloor c_k \rfloor$ and cost $c^1 = -1$. $e_2(R_k)$ has capacity $\lceil c_k \rceil - \lfloor c_k \rfloor$ and cost $c^2 = \lfloor c_k \rfloor + \lceil c_k \rceil - 2c_k$. For the third edge $e_3(R_k)$, its capacity is ∞ and its cost is $c^3 = 1$. Since $c^1 \leq c^2 \leq c^3$, to minimize the overall cost for these three edges the flow at $e_2(R_k)$ is zero unless the first edge $e_1(R_k)$ is full; that is, the flow at $e_1(R_k)$ is $\lfloor c_k \rfloor$. Similarly, flow at $e_3(R_k)$ is positive only if the two edges $e_1(R_k)$ and $e_2(R_k)$ are both full.

The cost associated with an edge is determined by multiplying the above coefficient to the flow in the edge. When the total amount of flow in the three edges is given by y_k, the total cost is given by $f_k(y_k) - c_k$ in any case (see, e.g., Ahuja, Magnanti, and Orlin [1]). Since c_k is a constant, the constant term does not affect the optimality.

Once a network is constructed, we can find an optimal rounding in time $O(|E| \log U(|E| + |V| \log |V|))$ for a network with node set V and edge set E and the largest integral capacity U, using the scaling algorithm by Edmonds and Karp [8]. In our case, $|V|$, $|E|$, and U are all $O(n)$, and thus we have $O(n^2 \log^2 n)$, where n is the number of matrix elements. □

THEOREM 4.2. *Given a* $[0, 1]$*-matrix* A *and a 2-laminar family* $\mathcal{F}$*, an optimal binary matrix* B *that minimizes the distance* $Dist_p^{\mathcal{F}}(A, B)$ *(*$p \geq 2$*) can be computed*

in $O(n^2 \log^3 n)$ time.

Proof. In this case, f_i is a piecewise-linear convex function with $O(n)$ break points. We apply the convex-cost flow algorithm [18]. We omit details. □

5. Upper bounds for the L_p discrepancy.

5.1. L_p discrepancy for a unimodular hypergraph. In this subsection, we prove the following theorem for the L_p discrepancy of a unimodular family.

THEOREM 5.1. *If $\mathcal{F}$ is unimodular and $p \leq 3$, for any $A \in \mathcal{A}$ we have*

$$\min_{B \in \mathcal{B}} Dist_p^{\mathcal{F}}(A, B) \leq \frac{1}{2}|\mathcal{F}|^{1/p}.$$

Proof. There exists $\hat{B} \in \mathcal{B}$ such that $|\hat{B}(R_i) - A(R_i)| \leq 1$ holds for any R_i. The existence of such $\hat{B}$ is known by Theorem 1.2. However, for completeness, we shall give the proof. Consider the problem (P2) and its continuous relaxation (P3). It is then obvious that

$$\begin{aligned} &b_{ij} = a_{ij} \text{ for every } i \text{ and } j \text{ and} \\ &y_k = A(R_k) \quad \text{for every} \quad R_k \in \mathcal{F} \end{aligned} \tag{5.1}$$

is a feasible solution to (P3). Now we add lower and upper bound constraints for each variable y_k:

$$\lfloor A(R_k) \rfloor \leq y_k \leq \lceil A(R_k) \rceil.$$

Notice that the addition of these constraints to (P3) maintains total unimodularity of the constraints. Let (P4) denote the problem (P3) with these constraints. Since $f_k(y_k)$ is a linear function in the interval $[\lfloor A(R_k) \rfloor, \lceil A(R_k) \rceil]$, (P4) is a linear program.

Since (P3) has a feasible solution satisfying all constraints of (P4), (P4) also has a feasible solution. Since (P4) is a linear program over totally unimodular constraints, its optimal solution is an integral solution, and the corresponding objective value gives an upper bound on the optimal objective value of (P2). Thus, the objective value for (P4) of the above defined feasible solution gives an upper bound on the optimal objective value of (P2).

Let us now estimate the upper bound on $f_k(y_k)$ at $y_k = A(R_k)$. Let $a = A(R_k) - \lfloor A(R_k) \rfloor$. We then have $f_k(\lfloor A(R_k) \rfloor) = a^p$ and $f_k(\lceil A(R_k) \rceil) = (1-a)^p$. Therefore,

$$f_k(A(R_k)) = a^p(1-a) + (1-a)^p a \tag{5.2}$$

holds. We can see $f_k(A(R_k)) \leq (1/2)^p$ holds if $p \leq 3$. Thus, the optimal objective value of (P4) is at most $(1/2)^p|\mathcal{F}|$. Since the optimal objective value of (P4) is an upper bound on that for (P2), $(1/2)^p|\mathcal{F}|$ gives an upper bound on the optimal objective value for (P2). □

It is easy to give an instance to show that the bound is tight: Consider Baranyai's problem on a matrix having $\frac{1}{2}$ entries in its diagonal position (other entries are zeros).

For the case $p > 3$, we have the following.

THEOREM 5.2. *If $\mathcal{F}$ is unimodular and $p > 3$, for any $A \in \mathcal{A}$ we have*

$$\begin{aligned} &\min_{B \in \mathcal{B}} Dist_p^{\mathcal{F}}(A, B) \\ &\leq (p^p/(p+1)^{p+1} + 2^p(p-1)/(p+1)^{p+1})^{1/p}|\mathcal{F}|^{1/p}. \end{aligned}$$

Proof. This follows from the fact that $a^p(1-a)+(1-a)^p a < p^p/(p+1)^{p+1} + 2^p(p-1)/(p+1)^{p+1}$ for $0 \le a \le 1$. □

The term $(p^p/(p+1)^{p+1} + 2^p(p-1)/(p+1)^{p+1})^{1/p}$ is 0.550 and 0.587 if $p=4$ and $p=5$, respectively, and it is always less than $p/(p+1)$.

5.2. Discrepancy bounds for the family of 2×2 regions. The method in the previous subsection does not work for a nonunimodular case. A simple but interesting family defining a nonunimodular hypergraph is the family of all 2×2 regions of A. The known upper bound is merely $\frac{5}{3}|\mathcal{F}|^{1/p}$ from the corresponding L_∞ result [3]. We obtain the following result.

THEOREM 5.3. *For any $A \in \mathcal{A}(G_{MN})$ and a family $\mathcal{F}$ of 2×2 regions of the matrix, we have*

$$\min_{B \in \mathcal{B}} Dist_1^{\mathcal{F}}(A,B) \le \frac{3}{4}|\mathcal{F}|.$$

Proof. Let us consider the matrix rounding problem $P = \mathcal{P}(G_{MN}, \mathcal{F}, 1)$ for the family $\mathcal{F}$ of all 2×2 regions. We define another problem $\hat{P}$ defined over another family $\hat{\mathcal{F}}$ of regions consisting of two tiles:

$$\mathcal{T}_1 = \{(i,j),(i+1,j)\} \text{ for } (i,j) \in G_{MN},$$
$$\mathcal{T}_2 = \{(i,j),(i+1,j),(i,j+1),(i+1,j+1)\} \text{ for } (i,j) \in G_{MN} \text{ and } i+j = \text{even}.$$

Let B^* and $\hat{B}$ denote the optimal binary matrix for P and $\hat{P}$, respectively.

We now show

$$\sum_{R \in \mathcal{F}} |A(R) - B^*(R)| \le 3|\mathcal{F}|/4.$$

Let (R_1, R_2) be a partition of a 2×2 region R into two disjoint 2×1 regions. Then for any A and B we have

$$\begin{aligned} &|A(R) - B(R)| \\ &\le |A(R_1) - B(R_1)| + |A(R_2) - B(R_2)|. \end{aligned} \tag{5.3}$$

Recall that $\mathcal{F}$ consists of all 2×2 regions and $\hat{\mathcal{F}}$ consists of constrained 2×2 regions and all 2×1 regions. Then $\mathcal{F} \backslash \hat{\mathcal{F}}$ consists of 2×2 regions that are not included in $\hat{\mathcal{F}}$, $\mathcal{F} \cap \hat{\mathcal{F}}$ consists of 2×2 regions that are included in $\hat{\mathcal{F}}$, and $\hat{\mathcal{F}} \backslash \mathcal{F}$ consists of all 2×1 regions. Now we have

$$\begin{aligned} &\sum_{R \in \mathcal{F}} |A(R) - B^*(R)| \\ &= \sum_{R \in \mathcal{F} \backslash \hat{\mathcal{F}}} |A(R) - B^*(R)| + \sum_{R \in \mathcal{F} \cap \hat{\mathcal{F}}} |A(R) - B^*(R)| \\ &\le \sum_{R \in \mathcal{F} \backslash \hat{\mathcal{F}}} |A(R) - \hat{B}(R)| + \sum_{R \in \mathcal{F} \cap \hat{\mathcal{F}}} |A(R) - \hat{B}(R)| \\ &\le \sum_{R \in \mathcal{F} \cap \hat{\mathcal{F}}} |A(R) - \hat{B}(R)| + \sum_{R \in \hat{\mathcal{F}} \backslash \mathcal{F}} |A(R) - \hat{B}(R)| \\ &\quad \text{(from (5.3))} \\ &= \sum_{R \in \hat{\mathcal{F}}} |A(R) - \hat{B}(R)|. \end{aligned} \tag{5.4}$$

Since $\hat{\mathcal{F}}$ is a 2-laminar family, its associated incidence matrix is totally unimodular. As in the proof of Theorem 5.1, we consider the linear program $\hat{Q}'$ which is a continuous relaxation of $\hat{P}$ with the additional constraints

$$\lfloor A(R_k) \rfloor \leq y_k \leq \lceil A(R_k) \rceil$$

for all $R_k \in \hat{\mathcal{F}}$. From the total unimodularity of the incidence matrix, there exists an optimal solution to $\hat{Q}'$ such that it is integral. Let $\hat{B}'$ denote such solution. For a feasible solution to $\hat{Q}'$ defined by

$$\begin{aligned} &b_{ij} = a_{ij} \text{ for every } i \text{ and } j \\ &\quad \text{and } y_k = A(R_k) \text{ for every } R_k \in \hat{\mathcal{F}}, \end{aligned} \tag{5.5}$$

its objective value gives an upper bound on the optimal objective value of $\hat{Q}'$ which in turn gives an upper bound on the optimal objective value of $\hat{P}$. Therefore, we have

$$\begin{aligned} \sum_{R_k \in \hat{\mathcal{F}}} |A(R_k) - \hat{B}(R_k)| &\leq \sum_{R_k \in \hat{\mathcal{F}}} |A(R_k) - \hat{B}'(R_k)| \\ &\leq \sum_{R_k \in \hat{\mathcal{F}}} f_k(A(R_k)). \end{aligned} \tag{5.6}$$

From Theorem 5.1, the rightmost term of (5.6) is bounded by $|\hat{\mathcal{F}}|/2$, which is almost equal to $3|\mathcal{F}|/4$ for a sufficiently large n. This completes the proof of the theorem. □

6. Application to digital halftoning. The quality of color printers has been drastically improved in recent years, mainly based on the development of the fine control mechanism. On the other hand, there seems to be no great invention on the software side of the printing technology. What is required is a technique to convert a continuous-tone image into a binary image consisting of black and white dots so that the binary image looks very similar to the input image. From a theoretical standpoint, the problem is how to approximate an input $[0,1]$-array by a binary array. Since this is one of the central techniques in computer vision and computer graphics, a great number of algorithms have been proposed (see, e.g., [13, 9, 4, 15, 17]). However, there have been very few studies toward the goal of achieving an optimal binary image under some reasonable criterion; maybe it is because the problem itself is very practically oriented. A desired output image is the one which looks similar to the input image to the human visual system. The most popular distortion criterion that is used in practice is perhaps frequency weighted mean square error (FWMSE) [16], which is defined by

$$W(G,X) = \sum_{(i,j) \in G_{MN}} \left[\sum_{k=-K}^{K} \sum_{l=-K}^{K} v_{|k||l|} a_{i+k,j+l} - \sum_{k=-K}^{K} \sum_{l=-K}^{K} v_{|k||l|} b_{i+k,j+l} \right]^2.$$

Here, $V = (v_{|k||l|})$, $-K \leq k, l \leq K$, is an impulse response that approximates the characteristics of the human visual system and K is some small constant, say 3. Our discrepancy measure which has been discussed in this paper is a hopeful replacement. Indeed, the L_2-discrepancy measure can be regarded as a simplified version of the FWMSE criterion.

We have implemented the algorithm using LEDA [14] functions for finding minimum-cost flow and applied it to several test images to compare its results with the error diffusion algorithm which is most commonly used in practice. The data we used for our experiments are *standard high precision picture data* created by the Institute of Image Electronics Engineers of Japan, which include four standard pictures called "Bride," "Harbor," "Wool," and "Bottles." They are color pictures of eight bits each in RGB. Their original picture size is 4096×3072. In our experiments we scaled them down to 1024×768 in order to shorten the running time of the program. Figure 3 shows experimental results for "Wool" to compare our algorithm with error diffusion. Our algorithm has been implemented using a 2-laminar family defined by the two tiles (b) and (c) depicted in Figure 4. We have used the L_1 measure. By our experience through experiments, it seems hard to have such a nice-looking output by the L_∞ measure.

FIG. 3. *Experimental results. Output images by the error diffusion algorithm (above) and the algorithm in this paper (below).*

7. Concluding remarks. We have considered the matrix rounding problem based on L_p-discrepancy measure. Although we have shown that the measure is useful in application to the digital-halftoning application, the current algorithm is too

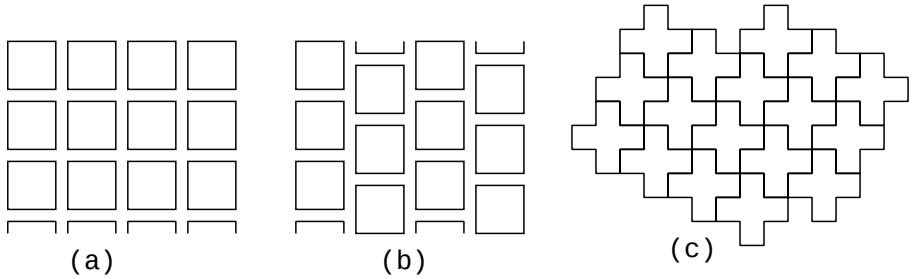


FIG. 4. *Three different partitions of the image plane* (a) *by* 2×2 *squares,* (b) *vertically shifted* 2×2 *squares, and* (3) *cross patterns consisting of five pixels.*

slow if we want to require speed together with the high-quality requirement. The problem comes from the quadratic time complexity. It is desired to design a faster algorithm (even an approximation algorithm). Moreover, it is an interesting question to investigate what kind of region families give the best criterion for the halftoning application. Once we know such a region family, it is valuable to design an algorithm (heuristic algorithm if the problem for solving the optimal solution is intractable) for the criterion.

Acknowledgments. The authors would like to thank Tomomi Matsui, Koji Nakano, and Hiroshi Nagamochi for their valuable comments and helpful discussions.

REFERENCES

[1] R. AHUJA, T. MAGNANTI, AND J. ORLIN, *Network Flows, Theory Algorithms and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] T. ASANO, T. MATSUI, AND T. TOKUYAMA, *Optimal roundings of sequences and matrices*, Nordic J. Comput., 7 (2000), pp. 241–256.

[3] T. ASANO AND T. TOKUYAMA, *How to color a checkerboard with a given distribution—Matrix rounding achieving low* 2×2*-discrepancy*, in Algorithms and Computation, Lecture Notes in Comput. Sci. 2223, Springer, Berlin, 2001, pp. 636–648.

[4] B. E. BAYER, *An optimum method for two-level rendition of continuous-tone pictures*, IEEE Int. Conf. Commun., 1 (1973), pp. 11–15.

[5] Z. BARANYAI, *On the factorization of the complete uniform hypergraphs*, in Infinite and Finite Sets, Colloq. Math. Soc. J'anos Bolyai 10, A. Hanaj, R. Rado, and V. T. Sós, eds., North-Holland, Amsterdam, 1975, pp. 91–108.

[6] J. BECK AND V. T. SÓS, *Discrepancy theory*, in Handbook of Combinatorics, Vol. II, R. Graham, M. Grötschel, and L. Lovász, eds., Elsevier, Amsterdam, 1995.

[7] B. DOERR, *Lattice approximation and linear discrepancy of totally unimodular matrices–Extended abstract*, in Proceedings of the 12th SIAM-ACM Symposium on Discrete Algorithms, Washington, D.C., 2001, pp. 119–125.

[8] J. EDMONDS AND R. M. KARP, *Theoretical improvements in algorithmic efficiency for network flow problems*, J. Assoc. Comput. Mach., 19 (1972), pp. 248–264.

[9] R. W. FLOYD AND L. STEINBERG, *An adaptive algorithm for spatial gray scale*, Journal of the Society for Information Display, 17 (1976), pp. 75–77.

[10] A. GHOULIA-HOURI, *Characterisation des matrices totalement unimodulaires*, C.R. Acad. Sci. Paris, 254 (1962), pp. 1192–1194.

[11] D. S. HOCHBAUM AND J. G. SHANTHIKUMAR, *Nonlinear separable optimization is not much harder than linear optimization*, J. Assoc. Comput. Mach., 37 (1990), pp. 843–862.

[12] A. V. KARZANOV AND S. T. MCCORMICK, *Polynomial methods for separable convex optimization in unimodular linear spaces with applications*, SIAM J. Comput., 26 (1997), pp. 1245–1275.

[13] D. E. KNUTH, *Digital halftones by dot diffusion*, ACM Trans. Graphics, 6 (1987), pp. 245–273.

[14] *The LEDA Homepage*, http://www.algorithmic-solutions.com/as_html/products/products.html.

[15] J. O. Limb, *Design of dither waveforms for quantized visual signals*, Bell System Tech. J., 48 (1969), pp. 2555–2582.

[16] Q. Lin, *Halftone Image Quality Analysis Based on a Human Vision Model*, Proc. SPIE 1913, SPIE, Bellingham, WA, 1993, pp. 378–389.

[17] B. Lippel and M. Kurland, *The effect of dither on luminance quantization of pictures*, IEEE Trans. Commun. Tech., COM-19 (1971), pp. 879–888.

[18] M. Minoux, *Solving integer minimum cost flows with separable cost objective polynomially*, Math. Programming Stud., 26 (1986), pp. 237–239.

[19] K. Sadakane, N. Takki-Chebihi, and T. Tokuyama, *Combinatorics and algorithms on low-discrepancy roundings of a real sequence*, in Procedings of the 28th International Colloquium on Automata, Languages and Programming, Heraklion, Crete, Greece, 2001, Lecture Notes in Comput. Sci. 2076, Springer, Berlin, 2001, pp. 166–177.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1436–1448

FINDING POINTS ON CURVES OVER FINITE FIELDS*

JOACHIM VON ZUR GATHEN†, IGOR SHPARLINSKI‡, AND ALISTAIR SINCLAIR§

Abstract. We solve two computational problems concerning plane algebraic curves over finite fields: generating a uniformly random point, and finding all points deterministically in amortized polynomial time (over a prime field, for nonexceptional curves).

Key words. finite fields, algebraic curves, algebraic geometry, computer algebra, probabilistic algorithms, random sampling

AMS subject classifications. Primary, 68W30; Secondary, 14Q05, 11Y40

DOI. 10.1137/S0097539799351018

1. Introduction.

Let q be a prime power, let $\mathbb{F}_q$ be a finite field with q elements, let $f \in \mathbb{F}_q[x, y]$ of total degree n, and let $\mathcal{C} = \{(a, b) \in \mathbb{F}_q^2 : f(a, b) = 0\} = \{f = 0\}$ be the plane curve defined by f. We consider two problems of finding points on this curve: probabilistically finding a uniformly distributed random point, and deterministically computing all its points.

Curves over finite fields play a role in several applications: factoring integers with elliptic curves, testing primality with elliptic curves (or more general algebraic varieties), algebro-geometric Goppa codes, and fast multiplication over finite fields. For these applications, special methods for finding points (if needed) are used. This paper presents the first general and systematic approach to the problem, to the authors' knowledge.

Throughout this paper, we will assume that f is squarefree and denote by σ the number of absolutely irreducible components of $\mathcal{C}$ which are defined over $\mathbb{F}_q$. The famous theorem of Weil says that the number of points $\#\mathcal{C}$ on $\mathcal{C}$ satisfies

$$| \#\mathcal{C} - \sigma q | \leq n^2 q^{1/2}. \tag{1.1}$$

The case of an *exceptional curve*, corresponding to $\sigma = 0$, needs special treatment and is dealt with in section 5. So for now we assume that $\sigma \geq 1$.

In section 2, we provide a polynomial-time solution for the probabilistic variant of our question: generating a uniform random point on $\mathcal{C}$. The algorithm is elementary and is based on the idea of rejection sampling. We also use this algorithm to obtain arbitrarily good probabilistic estimates of $\#\mathcal{C}$.

With deterministic methods, the "brute force" approach to computing all points on $\mathcal{C}$ via finding, for each $a \in \mathbb{F}_q$, all $b \in \mathbb{F}_q$ with $f(a, b) = 0$ takes $O^{\sim}(n^2 q^{3/2})$ operations in $\mathbb{F}_q$, using the fastest known deterministic algorithms to factor the univariate

*Received by the editors January 19, 1999; accepted for publication (in revised form) July 31, 2002; published electronically September 17, 2003. A preliminary version of this paper appeared in the *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science,* Milwaukee, WI, 1995, pp. 284–292.

http://www.siam.org/journals/sicomp/32-6/35101.html

†Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, 33095 Paderborn, Germany (gathen@upb.de). Part of this author's work was done while he was visiting Macquarie University and ICSI, Berkeley.

‡Department of Computing, Macquarie University, Sydney, NSW 2109, Australia (igor@comp.mq.edu.au). Part of this author's work was done while he was visiting Universität Paderborn.

§Computer Science Division, University of California Berkeley, Berkeley, CA 94720-1776 (sinclair@cs.berkeley.edu). The research of this author was supported in part by NSF grant CCR-9505448 and by the International Computer Science Institute.

polynomial $f(a,y)$ for all $a \in \mathbb{F}_q$ (Shoup (1990), section 1.1 of Shparlinski (1999), von zur Gathen and Shoup (1992)). We present in section 3 a deterministic method that uses $O^{\sim}(n^5 q)$ operations, i.e., polynomial time per point. The central tool for our estimates is a bound of Perel'muter (1969) on a certain exponential sum. In order to use this, we have to study in section 4 some geometric and arithmetic properties of the fiber square $\mathcal{C} \times_\pi \mathcal{C}$. Our approach works only in the case of a prime field $\mathbb{F}_q$, with $q = p$ prime, and does not work for exceptional curves.

Shoup (1990) has exhibited a deterministic univariate factoring algorithm which for almost all polynomials runs in polynomial time. Our deterministic result has two interpretations: the first is that the members of a "small" parametrized family $f(a,y)$ of univariate polynomials for all $a \in \mathbb{F}_p$ can be factored deterministically in (amortized) polynomial time. The second is that all points on a plane algebraic curve over $\mathbb{F}_p$ can be found deterministically in (amortized) polynomial time.

Finally, section 5 presents a discussion of the case of exceptional curves which has been excluded in the other sections.

A different set of results on our problem (and higher-dimensional varieties) was obtained by Adleman and Huang (2001), Huang and Wong (1999), Huang and Ierardi (1998), and Huang and Wong (1998).

2. Generating uniform random points. In order to generate random points on a plane curve, it is natural to take random points on a coordinate axis and compute points "above" them. So let $\pi\colon \mathcal{C} \to \mathbb{F}_q$ be the projection onto the first coordinate. For $0 \le i \le n$, let

$$R_i = \{a \in \mathbb{F}_q \colon \#\pi^{-1}(\{a\}) = i\}$$

be the set of points with exactly i preimages, and let $r_i = \#R_i$. We assume that $\mathcal{C}$ contains no vertical lines so that no $x-a$ with $a \in \mathbb{F}_q$ divides f. Then $\mathbb{F}_q = \bigcup_{0\le i\le n} R_i$ is a partition, and

$$q = \sum_{0\le j\le n} r_j, \quad \#\mathcal{C} = \sum_{1\le j\le n} j r_j.$$

Algorithm 2.1 (random point).
Input: $f \in \mathbb{F}_q[x,y]$ of degree n.
Output: Either a uniform random point (a,b) on $\mathcal{C} = \{f = 0\} \subseteq \mathbb{F}_q^2$, or "failure".

1. Choose $a \in \mathbb{F}_q$ uniformly at random.
2. Compute $f_a = \gcd(y^q - y, f(a,y)) \in \mathbb{F}_q[y]$.
3. Choose a random root $b \in \mathbb{F}_q$ of f_a. [Then $(a,b) \in \mathcal{C}$.]
4. Set $i = \deg f_a$. [Then $a \in R_i$.]
5. Choose YES with probability i/n, and NO with probability $1 - i/n$. If YES was chosen, return (a,b), and otherwise return "failure".

Theorem 2.2. *Suppose that $\mathcal{C}$ is a nonexceptional curve without vertical lines. Then the algorithm returns a uniform random point on $\mathcal{C}$ with probability*

$$\frac{\#\mathcal{C}}{nq} \ge \frac{1}{n}\left(1 - n^2 q^{-1/2}\right),$$

and "failure" with probability $1 - \#\mathcal{C}/nq$. For every $P \in \mathcal{C}$, P is returned with probability $1/nq$. The algorithm can be performed with an expected number of $O(n \log n \log(nq) \cdot \operatorname{loglog} n)$ operations in $\mathbb{F}_q$.

Proof. Let $P = (a, b) \in \mathcal{C}$ with $a \in R_i$. Then

$$\text{prob}\{P \text{ is returned}\} = \frac{1}{q} \cdot \frac{1}{i} \cdot \frac{i}{n} = \frac{1}{nq}.$$

We denote by $\mathsf{M}(n)$ a *multiplication time* so that the product of two polynomials in $\mathbb{F}_q[x]$ of degree at most n can be computed with $O(\mathsf{M}(n))$ operations in $\mathbb{F}_q$. Then we can take $\mathsf{M}(n) = n \log n \log\log n$, and a gcd can be computed with $O(\mathsf{M}(n) \log n)$ operations. Using repeated squaring to calculate $y^q \bmod f(a, y)$ with $O(\mathsf{M}(n) \log q)$ operations, the cost of step 2 is $O(\mathsf{M}(n) \log(nq))$. The polynomial f_a is a product of $i = \deg f_a$ many linear factors in $\mathbb{F}_q[x]$. If we find a root using the randomized algorithms of Cantor and Zassenhaus (1981), it will be uniformly randomly distributed among these i roots. The algorithm splits the polynomial recursively into two factors, one of which is $\gcd(y^{(q-1)/2} - 1, f_a(y + b))$ for a random $b \in \mathbb{F}_q$, and continues with the smaller factor. (For even q, a different formula is used.) We expect $O(\log i)$ splits to suffice, and each costs $O(\mathsf{M}(i) \log(qi))$ operations in $\mathbb{F}_q$. □

We think of q as being much larger than n, say, $q \geq c^2 n^4$ for some constant c. Then the success probability of Algorithm 2.1 is at least $\frac{1}{n}(1 - c^{-1})$. Of course, we can increase the success probability by repeated runs of the algorithm.

We can adapt Algorithm 2.1 to obtain an arbitrarily good approximation for $\#\mathcal{C}$, the number of points on $\mathcal{C}$. An (ϵ, δ)-*approximation* ρ to $\#\mathcal{C}$ satisfies

$$\text{prob}\ \{|\ \rho - \#\mathcal{C}\ | \leq \epsilon \#\mathcal{C}\} \geq 1 - \delta.$$

To achieve this, we simply run Algorithm 2.1 k times, count the number t of times that YES was chosen in step 5, and return the value $\rho = tnq/k$. Since YES is output with probability $\#\mathcal{C}/nq$, the expected value of ρ is exactly $\#C$, so it is an unbiased estimator. The unbiased estimator theorem of Karp, Luby, and Madras (1989) tells us how large k, the number of samples, should be to guarantee an (ϵ, δ)-approximation. This value is

$$k = \lceil 4\beta \log_e(2/\delta) \epsilon^{-2} \rceil, \tag{2.1}$$

where β is an upper bound on $nq/\#\mathcal{C}$. However, $nq/\#\mathcal{C} \leq n(1 - n^2 q^{-1/2})^{-1}$, so β is not very large. In fact, assuming as before that $q \gg n^4$, the number of samples required is only about $4n \log_e(2/\delta)\epsilon^{-2}$.

It is even easier in principle to estimate the individual r_i's. We choose k random values $a \in \mathbb{F}_q$, determine for each the j with $a \in R_j$, count the number t of times that $j = i$ occurred, and return the value $\rho_i = tq/k$. This is obviously an unbiased estimator of r_i, and the number of samples required for an (ϵ, δ)-approximation is as in (2.1), where now $\beta = \beta_i$ is an upper bound on q/r_i. With a parameter α, this implies that, by taking

$$k = \lceil 4\alpha n \log_e(2/\delta) \epsilon^{-2} \rceil,$$

we get an (ϵ, δ)-approximation for any r_i satisfying $r_i \geq q/\alpha$. Since

$$n \sum_{1 \leq i \leq n} r_i \geq \sum_{1 \leq i \leq n} i r_i = \#\mathcal{C},$$

the r_i's are on average at least $\#\mathcal{C}/n^2 \geq q(n^{-2} - q^{-1/2})$. Thus "on average" k will only be about $4n^2 \log_e(2/\delta)\epsilon^{-2}$, assuming as before that $q \gg n^4$. Such a value will

enable us to estimate the "large" r_i's though not, of course, the small ones. In fact, when q is large compared to n^{6n}, then the r_i separate into two classes: Lemma 2.3 of von zur Gathen and Shparlinski (1998) implies that either $r_i \geq \frac{q}{i!(n-i)!} - 2n^{2n}q^{1/2}$ is reasonably large or $r_i \leq 2n^{2n}q^{1/2}$ is very small. Of course, the "reasonably large" may still be very small, and about q/r_i samples are required. Thus if we use $\beta_i = n!$, then in the first case we obtain an (ϵ, δ)-approximation scheme for r_i, and in the second we expect to find no $a \in R_i$.

Since

$$\frac{\#\mathcal{C}}{n} \leq \sum_{1 \leq i \leq n} r_i \leq \sum_{1 \leq i \leq n} i r_i = \#\mathcal{C},$$

the r_i's are on average at least $\#\mathcal{C}/n^2$. To find approximations only to the "large" r_i's, we might use $\beta_i = \lambda n^2$, with some small number λ.

3. Deterministic construction of all points. In this section, we present a deterministic algorithm for finding all points on $\mathcal{C} = \{f = 0\}$ over a prime field $\mathbb{F}_p$. It employs a deterministic polynomial-time algorithm for finding all roots of the univariate polynomials $f(a, y)$, with $a \in \mathbb{F}_p$. This algorithm does not factor $f(a, y)$ completely for all a, but we show that there are only about $\sqrt{p}$ exceptional a, and for these we use an always successful deterministic algorithm with time about $\sqrt{p}$; thus the total time is proportional to p, which is about the size of $\mathcal{C}$. Everything is polynomial in the degree n.

As a first step, we factor f into irreducible factors in $\mathbb{F}_p[x, y]$. The bivariate factoring algorithms (Lenstra (1985), von zur Gathen (1984), von zur Gathen and Kaltofen (1985)) can actually be made into deterministic reductions from bivariate to univariate factorization over finite fields. Thus f can be factored with $n^{O(1)}p^{1/2}$ operations in $\mathbb{F}_p$. From now on, we assume that f is irreducible.

The projection $\pi : \mathcal{C} = \{f = 0\} \to \mathbb{F}_p$ onto the first coordinate is called *separable* if and only if $h_y = \partial h/\partial y \neq 0$ for each irreducible factor $h \in \mathbb{F}_p[x, y]$ of f. A simple example of an inseparable projection is given by $f = x - y^p \in \mathbb{F}_p[x, y]$. The curve $\mathcal{C} = \{x = y^p\}$ is smooth, and all tangents to $\mathcal{C}$ are vertical.

Let $\varphi: \mathbb{F}_p \to \mathbb{F}_p$ denote the absolute Frobenius map, with $\varphi(a) = a^p$. For our algorithms, it is convenient to have π separable, and the next lemma describes a simple procedure for achieving this by factoring out φ. (It actually works over any finite field of characteristic p.)

LEMMA 3.1. *Let $f \in \mathbb{F}_p[x, y]$ be irreducible. We can compute in polynomial time $g \in \mathbb{F}_p[x, y]$ and an integer $k \leq \log_p(\deg_y f)$ such that*

$$\mathrm{id} \times \varphi^k: \mathbb{F}_p^2 \longrightarrow \mathbb{F}_p^2$$

gives a bijection between $\{f = 0\}$ and $\{g = 0\}$, $\deg_x g = \deg_x f, \deg_y g \leq \deg_y f$, and $\pi: \{g = 0\} \to \mathbb{F}_p$ is separable.

Proof. We write $f = \sum_{i,j} f_{ij} x^i y^j$, with each $f_{ij} \in \mathbb{F}_p$. Then

$$f_y = 0 \Longleftrightarrow \forall i, j \ (f_{ij} \neq 0 \Rightarrow p \mid j).$$

If $f_y = 0$ and

$$h = \sum_{\substack{i,j \\ p \mid j}} f_{ij} x^i y^{j/p} \in \mathbb{F}_p[x, y],$$

then $f(a,b) = h(a,b^p)$ for all $(a,b) \in \mathbb{F}_p^2$, and thus $\mathrm{id} \times \varphi: \mathbb{F}_p^2 \to \mathbb{F}_p^2$ gives a bijection between $\{f=0\}$ and $\{h=0\}$. Furthermore, h is irreducible. We repeat this process until we obtain a polynomial $g \in \mathbb{F}_p[x,y]$ and $k \in \mathbb{N}$ with $g_y \neq 0$ and $\mathrm{id} \times \varphi^k$ a bijection between $\{f=0\}$ and $\{g=0\}$. □

ALGORITHM 3.2 (finding all points).
Input: $f \in \mathbb{F}_p[x,y]$ of degree n, where p is a prime.
Output: A list of all points $(a,b) \in \mathbb{F}_p^2$ with $f(a,b) = 0$.

1. Set $h = 288n^4\lceil \log_2 p \rceil^2$.
2. For all $a \in \mathbb{F}_p$
3. Compute $f_a = f(a,y) \in \mathbb{F}_p[y]$.
4. Compute $f_a^* = \gcd(y^p - y, f_a) \in \mathbb{F}_p[y]$.
5. For $0 \le t < h$ compute the two factors

$$g_{a,t} = \gcd\big((y-t)^{(p-1)/2} - 1, f_a^*\big),\ g_{a,t}^* = \gcd(y-t, f_a^*) \in \mathbb{F}_p[y]$$

of f_a^*.
6. Compute the common refinement of the partial factorizations from step 5.
7. If step 6 returns only linear factors $y-b$, then add all these (a,b) to the list. Otherwise, completely factor f_a^* with the deterministic algorithm of von zur Gathen and Shoup (1992), and add all resulting (a,b) to the list.

THEOREM 3.3. *Let p be a prime, $f \in \mathbb{F}_p[x,y]$ squarefree and nonexceptional, and $\pi: \mathcal{C} = \{f=0\} \to \mathbb{F}_p$ separable. Then the algorithm correctly computes all points on $\mathcal{C}$. It uses*

$$O(n^5 p \log n \operatorname{loglog} n \log(np) \log^2 p)$$

or $O^{\sim}(n^5 p)$ operations in $\mathbb{F}_p$.

Proof. For all $a, b \in \mathbb{F}_p$, we have

$$f(a,b) = 0 \iff f_a^*(b) = 0 \iff y - b \mid f_a^*.$$

Since step 7 returns all linear factors of f_a^*, the final list correctly contains all points of $\mathcal{C} = \{f=0\}$.

It remains to analyze the running time. The crucial point is to understand when step 6 succeeds in completely factoring f_a^*. Denote by $S \subseteq \mathbb{F}_p$ the set of all a for which this is not the case, and $s = \#S$. Furthermore, $\mathcal{C}_a = \pi_2(\mathcal{C} \cap (\{a\} \times \mathbb{F}_p))$ consists of all $b \in \mathbb{F}_p$ with $(a,b) \in \mathcal{C}$. Thus

$$S = \{a \in \mathbb{F}_p : \exists b, c \in \mathcal{C}_a\ b \neq c; b, c \ge h,\ \text{and } \forall t < h\ (y-b \mid g_{a,t} \iff y-c \mid g_{a,t})\}.$$

The refinement cost in step 6, if done along a binary tree, is $O(\mathsf{M}(n)\log n)$ for each t, or $O(h\,\mathsf{M}(n)\log n)$ in total. For $a \in S$, an application of the algorithm from von zur Gathen and Shoup (1992) costs $O(\mathsf{M}(n)\, p^{1/2} \log(np))$ operations in $\mathbb{F}_p$. The gcds in steps 4 and 5 are computed by repeated squaring for the required power of y and $y-t$, reducing after each multiplication modulo f_a and f_a^*, respectively.

For each a in step 2, we find the following number of operations in $\mathbb{F}_p$:

- step 3: $O(n^2)$,
- step 4: $O(\mathsf{M}(n)\log(np))$,
- step 5: $O(h\,\mathsf{M}(n)\log(np))$,
- step 6: $O(h\,\mathsf{M}(n)\log n)$,

∘ step 7: 0 if $a \in \mathbb{F}_p \setminus S$, and $O(\mathsf{M}(n)\, p^{1/2} \log(np))$ if $a \in S$.

The total cost is

$$O\big(p \cdot (n^2 + n^4 \mathsf{M}(n) \log(np) \log^2 p) + s\, \mathsf{M}(n) p^{1/2} \log(np)\big) \tag{3.1}$$

operations, and we now show that s is $O(n^2(n^2 + \log p)p^{1/2})$. This will imply the claim about the running time. We let

$$Q = \{u \in \mathbb{F}_p^\times : \exists v \in \mathbb{F}_p^\times \ u = v^2\} = \{u \in \mathbb{F}_p^\times : u^{(p-1)/2} = 1\}$$

be the set of nonzero squares in $\mathbb{F}_p$ and χ the quadratic character on $\mathbb{F}_p$, with

$$\chi(b) = \begin{cases} 1 & \text{if } b \in Q, \\ -1 & \text{if } b \notin Q,\ b \neq 0, \\ 0 & \text{if } b = 0. \end{cases}$$

For the time being, we work with an arbitrary integer parameter h; only at the end will we substitute the value from step 1. Set $H = \{0, \dots, h-1\} \subseteq \mathbb{F}_p$, where we identify $\mathbb{F}_p$ with $\{0, \dots, p-1\}$. Two distinct elements $b, c \in \mathbb{F}_p$ are *h-separated* if and only if $\chi(b-t) \neq \chi(c-t)$ for some $t \in H$. A set $B \subseteq \mathbb{F}_p$ is *h-separated* if any two distinct elements of B are. With this notation, we have, for $a \in \mathbb{F}_p$,

$$a \in S \Longrightarrow \mathcal{C}_a \text{ is not } h\text{-separated}.$$

The reverse implication is true if the non-h-separated $b, c \in \mathcal{C}_a$ are both at least h. If $a \in S$, then, for at least one pair of distinct elements $b, c \in \mathcal{C}_a$,

$$h = \sum_{0 \le t < h} \chi\big((t-b)(t-c)\big).$$

Now we let $k \in \mathbb{N}$ and

$$\begin{aligned} w &= \sum_{a \in \mathbb{F}_p} \sum_{\substack{b,c \in \mathcal{C}_a \\ b \neq c}} \Big| \sum_{0 \le t < h} \chi\big((t-b)(t-c)\big)\Big|^{2k} \\ &= \sum_{0 \le t_1, \dots, t_{2k} < h} \ \sum_{a \in \mathbb{F}_p} \sum_{\substack{b,c \in \mathcal{C}_a \\ b \neq c}} \chi\big((t_1-b)(t_1-c)\cdots(t_{2k}-b)(t_{2k}-c)\big). \end{aligned}$$

Then, by the above, $sh^{2k} \le w$. We consider the set

$$\mathcal{D}_0 = \{(a,b,c) \in \mathbb{F}_p^3 : f(a,b) = f(a,c) = 0, b \neq c\} \subseteq \mathbb{F}_p^3.$$

The fiber product $\mathcal{D} = \mathcal{C} \times_\pi \mathcal{C}$ is the closure of $\mathcal{D}_0$ in $\mathbb{F}_p^3$; it has degree at most $n(n-1) < n^2$ and is discussed in detail in section 4. Then

$$w = \sum_{t \in H^{2k}} \sum_{P \in \mathcal{D}} \chi\big(\psi_t(P)\big),$$

where the inner sum is over all $\mathbb{F}_p$-rational points $P = (a,b,c) \in \mathcal{D}$ with $b \neq c$, ψ_t is the polynomial

$$\psi_t = (y - t_1)\cdots(y - t_{2k})(z - t_1)\cdots(z - t_{2k}) \in \mathbb{F}_p[y,z]$$

in indeterminates y and z, and $\psi_t((a,b,c))$ is obtained by substituting b and c for y and z, respectively. Theorem 4.4 says that there are at most $(12kn^2h^{1/2})^{2k}$ values of $t \in H^{2k}$ for which $\rho(\psi_t)$ is a square in the global ring $\mathcal{O}_\mathcal{A}$ of some irreducible component $\mathcal{A} \subseteq \mathbb{F}^3$ of $\mathcal{D}$, where $\rho\colon \mathbb{F}[x,y,z] \longrightarrow \mathcal{O}_\mathcal{A}$ is the restriction map.

For other vectors $t \in \mathbb{F}^{2k}$, we may apply the bound on character sums along a curve from Perel'muter (1969) that gives

$$\sum_{P \in \mathcal{D}} \chi\big(\psi_t(P)\big) \le d \cdot \big(n^2(n^2+2k)p^{1/2}\big) \tag{3.2}$$

for some constant d. Perel'muter's bound holds for each irreducible component of $\mathcal{D}$; we also use the fact that no such component is vertical (Lemma 3.1 of von zur Gathen, Karpinski, and Shparlinski (1996)). Since their degrees sum to $\deg \mathcal{D} < n^2$, (3.2) follows. Therefore,

$$w \le (12kn^2h^{1/2})^{2k}p + d \cdot n^2(n^2+2k)h^{2k}p^{1/2},$$

$$s \le (12kn^2h^{-1/2})^{2k}p + d \cdot n^2(n^2+2k)p^{1/2}.$$

Now, using $k = \lceil \log_2 p \rceil$ and h as in step 1 of Algorithm 3.2, we find

$$(12kn^2h^{-1/2})^{2k} \le 2^{-k} \le 2^{-\log_2 p} = p^{-1}.$$

Hence

$$s = O\big(n^2(n^2+\log p)p^{1/2}\big).$$

Together with (3.1), this proves the estimate of the total cost. □

4. Squares on the fiber product. The goal of this section is to bound the number of products Ψ_t which are squares on some irreducible component of $\mathcal{D}$; this was used in the previous proof.

Let $\mathbb{F}$ be an algebraically closed field, let $f \in \mathbb{F}[x,y]$ be squarefree of degree $n \ge 1$, let $\mathcal{C} = \{f = 0\} \subseteq \mathbb{F}^2$ be the associated plane curve, and let $\pi\colon \mathcal{C} \longrightarrow \mathbb{F}$ be the first projection. We assume that π is separable. Then $\mathcal{D} = \mathcal{C} \times_\pi \mathcal{C} \subseteq \mathbb{F}^3$, the fiber square over π, can be defined as the closure in $\mathbb{F}^3$ of

$$\mathcal{D}_0 = \{(a,b,c) \in \mathbb{F}^3 \colon f(a,b) = f(a,c) = 0, b \ne c\}.$$

Furthermore, let $g = (f(x,y) - f(x,z))/(y-z) \in \mathbb{F}[x,y,z]$.

A smooth point $P = (a,b) \in \mathcal{C}$ is *critical* for π if and only if the tangent line $T_{P,\mathcal{C}}$ in $\mathbb{F}^2$ is vertical, as illustrated in Figure 4.1. If f is irreducible, this is equivalent to $f_y(a,b) = 0$, where $f_y = \partial f/\partial y \in \mathbb{F}[x,y]$; in general, we have to replace f by its (unique) irreducible factor on whose component P lies. Since π is separable, $\mathcal{C}$ has only finitely many critical points.

THEOREM 4.1. *Let $f \in \mathbb{F}[x,y]$ be squarefree, and let π be separable.*

(i) $\mathcal{D} = \{f(x,y) = g(x,y,z) = 0\}$.

(ii) $\mathcal{D} = \mathcal{D}_0 \cup \{(a,b,b)\colon (a,b) \in \mathcal{C}$ *is singular or critical*$\}$.

(iii) $(a,b,c) \in \mathcal{D}$ *with $b \ne c$ is singular on $\mathcal{D}$ if and only if either (a,b) or (a,c) is singular on $\mathcal{C}$, or both (a,b) and (a,c) are critical on $\mathcal{C}$. All points of $\mathcal{D} \setminus \mathcal{D}_0$ are singular on $\mathcal{D}$.*

(iv) $\deg \mathcal{D} \le n(n-1) < n^2$.

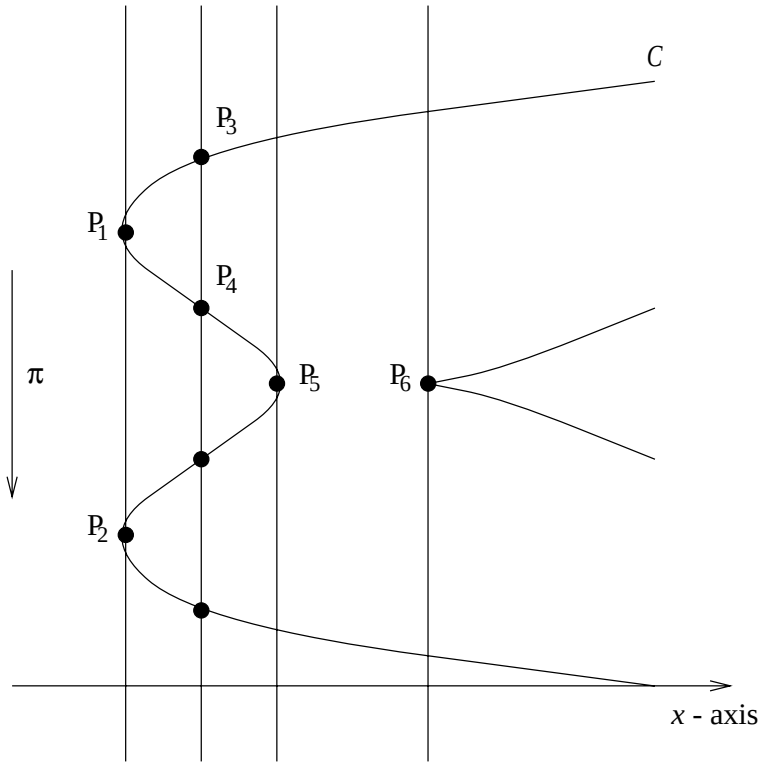


FIG. 4.1. *P_1, P_2, P_5 are critical for π, and P_6 is singular on $\mathcal{C}$. If $P_i = (a_i, b_i)$, then $(a_i, b_i, b_i) \in \mathcal{D} \cap \Delta$ for $i = 1, 2, 5, 6$. These four points are singular on $\mathcal{D}$. Furthermore, $(a_1, b_1, b_2) \in \mathcal{S} \subseteq \mathcal{D}$, and $(a_3, b_3, b_4) \in \mathcal{D} \setminus \mathcal{S}$.*

Proof. Let

$$\Delta = \{(a, b, b) \in \mathbb{F}^3 : a, b \in \mathbb{F}\}, \quad \mathcal{D}_1 = \{f(x, y) = g(x, y, z) = 0\}$$

so that Δ is the diagonal. Clearly $\mathcal{D} \setminus \Delta = \mathcal{D}_0$, and $\mathcal{D}_0 \subseteq \mathcal{D}_1$. By definition, $\mathcal{D}$ is the closure of $\mathcal{D}_0$, and thus $\mathcal{D} \subseteq \mathcal{D}_1$. We prove in the following that (ii) is valid with $\mathcal{D}_1$ instead of $\mathcal{D}$. Thus $\mathcal{D}_1 \cap \Delta$ is finite, and $\mathcal{D} = \mathcal{D}_1$ follows; this implies (i), (ii), and (iv).

So let u, v be indeterminates over $\mathbb{F}[x, y]$. Then the Taylor expansion of f around (u, v) of order 1 is

$$f(x, y) = f(u, v) + f_x(u, v)(x - u) + f_y(u, v)(y - v) + h$$

in $\mathbb{F}[x, y, u, v]$, with some $h \in (x - u, y - v)^2$. Therefore,

$$\begin{aligned} &g(x, y, z) \\ &= \frac{1}{y - z} \cdot (f_y(u, v)(y - v) - f_y(u, v)(z - w) + h(x, y, u, v) - h(x, z, u, v)) \\ &= f_y(u, v) + H, \end{aligned}$$

with some $H \in (x - u, y - v, z - v)$. Thus, for $(a, b) \in \mathcal{C}$,

$$(a, b, b) \in \mathcal{D}_1 \iff f_y(a, b) = 0 \iff (a, b) \text{ is singular or critical on } \mathcal{C}.$$

For (iii), let $(a,b,c) \in \mathcal{D}$ with $b \neq c$. The Jacobian of $\mathcal{D}$ at (a,b,c) is

$$J(a,b,c) = \begin{pmatrix} f_x(a,b) & \dfrac{f_x(a,b) - f_x(a,c)}{b-c} \\ f_y(a,b) & \dfrac{f_y(a,b)}{b-c} \\ 0 & \dfrac{-f_y(a,c)}{b-c} \end{pmatrix}.$$

After multiplying the second column by $c-b$ and then adding the first column to the second, we obtain the matrix

$$A = \begin{pmatrix} f_x(a,b) & f_x(a,c) \\ f_y(a,b) & 0 \\ 0 & f_y(a,c) \end{pmatrix}.$$

Thus

$$\begin{aligned} &(a,b,c) \text{ is singular on } \mathcal{D} \iff \text{rank } (J(a,b,c)) \leq 1 \\ &\iff \text{rank } (A) \leq 1 \\ &\iff (a,b) \text{ or } (a,c) \text{ is singular on } \mathcal{C}, \text{ or both are critical on } \mathcal{C}. \qquad \square \end{aligned}$$

The condition that π be separable is necessary since otherwise all points on $\mathcal{C}$ are critical. Recall the example $\mathcal{C} = \{x = y^p\}$, where $p = \operatorname{char} \mathbb{F}$, from section 3. Then $f_y = 0$, $\mathcal{C}$ is smooth, and all tangent lines to $\mathcal{C}$ are vertical. Furthermore, $\mathcal{D}_0 = \emptyset$, $g = (y^p - z^p)/(y-z) = (y-z)^{p-1}$, and $\mathcal{C} \times_\pi \mathcal{C}$ equals $\{(a,b,b) \in \mathbb{F}^3 : a = b^p\}$, counted $p-1$ times. On the other hand, when $\mathcal{C} = \{y = g(x)\}$ is the graph of a polynomial $g \in \mathbb{F}_q[x]$, then π is separable, and $\mathcal{D} = \emptyset$.

We define

$$\mathcal{S} = \{(a,b,c) \in \mathcal{D} : (a,b) \text{ or } (a,c) \text{ is singular or critical on } \mathcal{C}\}.$$

We now let $\mathcal{A}$ be an irreducible component of $\mathcal{D}$, and we want to estimate the number of t such that

$$\psi_t = \prod_{1 \leq i \leq 2k} (t_i - y)(t_i - z)$$

is a square in $\mathcal{O}_\mathcal{A}$. We let $\rho : \mathbb{F}[x,y,z] \longrightarrow \mathcal{O}_\mathcal{A}$ be the restriction map.

Let $t \in \mathbb{F}^{2k}$, and $T = \{1, \ldots, 2k\}$. The overall goal of this section is to show in Theorem 4.4 that only a few $\rho(\psi_t)$ are squares when t is chosen from a finite subset H of $\mathbb{F}^{2k}$. For a simple example of a square, we take the parabola $f = x - y^2$ so that $\mathcal{C} = \{x = y^2\}$, and $\mathcal{D} = \{x - y^2 = y + z = 0\}$ is irreducible. If $k = 1$ and $t_2 = -t_1$, then

$$\rho(\psi_t) = \rho((t_1 - y)(t_1 - z)(t_2 - y)(t_2 - z)) = \rho((t_1 - y)^2 (t_1 + y)^2) \tag{4.1}$$

is a square on $\mathcal{D}$.

The condition that $\rho(\psi_t)$ not be a square for (3.2) to hold is not an artifact of Perel'muter's proof, but without it (3.2) may actually fail to be true.

In what follows, we define several combinatorial objects on the index set T. We first collect pairs of equal values of t_i in a systematic way. Namely, we take the

lexicographically first maximal matching on the directed graph with vertex set T, and where (i,j) are connected if and only if $i<j$ and $t_i=t_j$. Then $T_1\subseteq T$ is defined as the set of these first coordinates i, and $\tau_1\colon T_1\to T$ is defined by $\tau_1(i)=j$ if (i,j) occurs in that matching. As an example, if $t_3=t_5=t_8=t_{11}=t_{13}$ and no other t_i equals these, then $T_1=\{3,5\}$, $\tau_1(3)=8$, and $\tau_1(5)=11$.

Next, we set

$$T_2=\{i\in T\setminus(T_1\cup\tau_1(T_1))\colon \mathcal{A}\cap\{y=t_i\}\subseteq\mathcal{S} \text{ or } \mathcal{A}\cap\{z=t_i\}\subseteq\mathcal{S}\}.$$

Then the t_i for

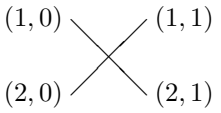
$$i\in T_3=T\setminus(T_1\cup\tau_1(T_1)\cup T_2)$$

are pairwise distinct, and $(T_1,\tau_1(T_1),T_2,T_3)$ is a partition of T. Next, we let

$$S_0=T_3\times\{0\},\quad S_1=T_3\times\{1\}$$

be two disjoint copies of T_3, and we now define a bipartite undirected graph $G=(S_0\cup S_1,E)$ as follows. For $i,j\in T_3$, $(i,0)$ and $(j,1)$ are connected in G if and only if there is some $(a,b,c)\in\mathcal{A}\setminus\mathcal{S}$ such that $b=t_i$ and $c=t_j$.

In the example (4.1) of a parabola, we have $T_1=T_2=\emptyset$, and

$G\ =$ 

.

LEMMA 4.2. *If $t\in\mathbb{F}^{2k}$ is such that $\rho(\psi_t)\in\mathcal{O}_{\mathcal{A}}$ is a square, then each vertex in G has degree at least one.*

Proof. By symmetry, it is sufficient to show the claim for a vertex $(i,0)\in S_0$.

Since $i\notin T_2$, we can choose some $P=(a,t_i,c)\in\mathcal{A}\setminus\mathcal{S}$; then $c\neq t_i$. Let

$$U_0=\{j\in T\colon t_j=t_i\},\quad U_1=\{j\in T\colon t_j=c\},$$

$\rho\colon\mathbb{F}[x,y,z]\to\mathcal{O}_{\mathcal{A}}$ be the restriction to $\mathcal{A}$, $\mathcal{R}=\mathcal{O}_{P,\mathcal{A}}$ be the local ring at P, which is a unique factorization domain (see, e.g., Shafarevich (1974), Theorem II.3.2), and $\lambda=(\mathcal{O}_{\mathcal{A}}\to\mathcal{O}_{P,\mathcal{A}})\circ\rho$ be the composition of ρ with the localization at P. Then $i\in U_0$ and $U_0,U_1\subseteq T\setminus T_2$.

For every $j\in T\setminus(U_0\cup\tau_1(U_0)\cup\{i\})$, we have $t_j\neq t_i$, and thus $\lambda(y-t_j)$ is a unit in $\mathcal{R}$. Similarly, each $\lambda(z-t_j)$ with $t_j\neq c$ is a unit in $\mathcal{R}$. Since $(a,t_i)\in\mathcal{C}$ is not critical for π, we have $f_y(a,t_i)\neq 0$, and therefore $\lambda(y-t_i)\in\mathcal{R}$ is a local parameter in $\mathcal{R}$. Similarly, each $\lambda(z-t_j)$ with $t_j=c$ is a local parameter in $\mathcal{R}$.

By the above, there is a unit $u\in\mathcal{R}$ such that

$$\begin{aligned}\lambda(\psi_t)&=\prod_{j\in T}\lambda(y-t_j)\cdot\prod_{j\in T}\lambda(z-t_j)\\&=u\cdot\prod_{j\in U_0\cup\tau_1(U_0)\cup\{i\}}\lambda(y-t_j)\cdot\prod_{j\in U_1}\lambda(z-t_j)\end{aligned}$$

is a square in $\mathcal{R}$. Thus the total number of local parameters in the product is even. We have $\#U_0=\#\tau_1(U_0)$ and $i\notin U_0\cup\tau_1(U_0)$. It follows that, in the left-hand product, the number of local parameters is odd, and therefore the same is true for the

right-hand product. Thus there exists some $j \in T_3$ with $t_j = c$; then $\{(i,0),(j,1)\} \in E$. ∎

We now take a maximal "disjoint" matching (V_0, V_1) in G of the following type. The sets $V_0, V_1 \subseteq T_3$ are disjoint, G induces a perfect matching on $(V_0 \times \{0\}) \cup (V_1 \times \{1\})$, and this matching is maximal. Furthermore, let $\mu : V_0 \longrightarrow V_1$ be the corresponding bijection, with $\mu(i) = j$ if and only if $\{(i,0),(j,1)\}$ occurs in the matching.

For every $i \in V_2 = T_3 \setminus (V_0 \cup V_1)$, $(i,0)$ is connected to some $(j,1) \in T_3 \times \{1\}$, and by the maximality of the matching, we have $j \in V_0 \cup V_1$. We take $\mu\colon V_2 \longrightarrow V_0 \cup V_1$ such that $\mu(i) = j$ for some such j and note that (V_0, V_1, V_2) is a partition of T_3.

Finally, we indicate how to describe t_i for $i \in V_0$ succinctly if $\{(i,0),(j,1)\} \in E$ and t_j is known. For this, we take an arbitrary total order $\prec$ on $\mathbb{F}$. For each $t \in \mathbb{F}$, $\mathcal{C} \cap \{y = t\}$ has at most n points, say, $(a_1,t),\dots,(a_l,t)$ with $l \le n$ and $a_1 \prec \cdots \prec a_l$. If $j = \mu(i)$ and $t = t_j$, then $(a_r, t_i, t_j) \in \mathcal{D} \setminus \mathcal{S}$ for one of those points, with $1 \le r \le l$. We choose the smallest such r; then $\mathcal{C} \cap \{x = a_r\}$ consists again of at most n points. We let v be the position of (a_r, t_i) in this list, ordered according to $\prec$, and set $\tau_3(i) = (r,v)$. Then t_i is determined by $j = \mu(i)$, t_j, and $\tau_3(i)$.

Similarly, we define $\tau_3\colon V_2 \longrightarrow \{1,\dots,n\}^2$ so that, for $i \in V_2$, t_i is determined by $j = \mu(i)$, t_j, and $\tau_3(i)$.

We have thus associated the following data to any $t \in \mathbb{F}^{2k}$ with $\rho(\psi_t)$ a square:

$$T_1, \tau_1, T_2, V_0, \mu, \tau_3, \text{ and } t_i \text{ for } i \in T_1 \cup T_2 \cup V_1. \tag{4.2}$$

LEMMA 4.3. *If $\rho(\psi_t)$ is a square in $\mathcal{O}_{\mathcal{A}}$, then t is determined by the data in* (4.2).

Proof. $(T_1, \tau_1(T_1), T_2, V_0, V_1, V_2)$ is a partition of T, and $t_i = t_{\tau_2(i)}$ for each $i \in T_1$. Thus it remains to show that each t_i with $i \in V_0 \cup V_2$ is determined by (4.2). However, that is precisely what the construction of μ and τ_3 achieves. ∎

We are now ready for the main result of this section, an upper bound on the number of ψ_t which are squares. The bound is rather coarse but sufficient for our purposes.

THEOREM 4.4. *Let $\mathbb{F}$ be an algebraically closed field, let $f \in \mathbb{F}[x,y]$ be squarefree, let $\mathcal{C} = \{f = 0\}$ with $\pi\colon \mathcal{C} \to \mathbb{F}$ be separable, let $H \subseteq \mathbb{F}$ be a finite set with h elements, and let $k \in \mathbb{N}$ be positive. The number of $t \in H^{2k}$ such that $\rho(\psi_t)$ is a square in $\mathcal{O}_{\mathcal{A}}$ for some irreducible component $\mathcal{A}$ of $\mathcal{C} \times_\pi \mathcal{C}$ is at most $(12kn^2h^{1/2})^{2k}$.*

Proof. We first fix a component $\mathcal{A}$ of $\mathcal{D}$ and show the corresponding bound. By Lemma 4.3, it is sufficient to give an upper bound on the number of choices for the data in (4.2).

The six sets $T_1, \tau_1(T_1), T_2, V_0, V_1, V_2$ form a partition of T, and there are at most 6^{2k} choices for this partition.

Suppose that these sets are chosen, with cardinalities $c_1, c_2, c_3, c_4, c_5, c_6$, respectively. Then $c_1 = c_2$, $c_3 < n^2$, and $c_4 = c_5$. The number of choices for τ_1 is at most $(2k)^{c_1}$, for μ at most $(2k)^{c_4+c_6}$, for τ_3 at most $(n^2)^{c_4+c_6}$, and for all t_i's required in (Theorem 4.1) at most $h^{c_1+c_5} \cdot (n^2)^{c_3}$. Since $c_1 + c_5 \le 2k/2 = k$, the total comes to

$$m = 6^{2k} \cdot (2k)^{c_1+c_4+c_6} \cdot (n^2)^{c_3+c_4+c_6} \cdot h^{c_1+c_5}. \tag{4.3}$$

Since $\deg \mathcal{D} \le n(n-1)$ by Theorem 4.1 (i), $\mathcal{D}$ has at most $n(n-1) < n^2$ irreducible components. So the total number of t considered is at most $n^2 m$, and

$$n^2 m \le 6^{2k} \cdot (2k)^{2k} \cdot (n^2 h^{1/2})^{2k}.$$

Here we use that either $c_1 + c_2 + c_5 > 0$ and then $n^2 \cdot (n^2)^{c_3+c_4+c_6} \le (n^2)^{2k}$, or $c_2 + c_3 + c_4 + c_6 > 0$ and then $n^2 (h)^{c_1+c_5} \le h^k$. ∎

5. Exceptional polynomials. In this section, we deal with the somewhat troublesome case excluded so far: exceptional polynomials, for which $\sigma = 0$. No analogue of the deterministic result of Theorem 3.3 is known for them, while the probabilistic results of section 2 carry over easily.

We first note that it is not surprising that they are difficult to deal with since any subset of $\mathbb{F}_q^2$ is an exceptional curve. If $c \in \mathbb{F}_q$ is a nonsquare and $f = x^2 + cy^2$, then f is exceptional and

$$\{f = 0\} = \{(0,0)\}, \tag{5.1}$$

and by translation and finite unions the claim follows. If char $\mathbb{F}_q \geq 3$, then (5.1) also holds for $f = x^{q-1} + y^{q-1}$. If $b \in \mathbb{F}_{q^2} \backslash \mathbb{F}_q$ with $b^2 \in \mathbb{F}_q$, then $b^{q-1} = (b^2)^{(q-1)/2} = -1$. Thus f is the product of all $x - by$ with these b, and thus f is exceptional, too.

Now, given an arbitrary $f \in \mathbb{F}_q[x,y]$ of degree n, there are well-known probabilistic algorithms with time polynomial in $n \log q$ that factor f into its irreducible factors over $\mathbb{F}_q$ (von zur Gathen and Kaltofen (1985)) and test each such factor for absolute irreducibility (Kaltofen (1985)). For simplicity, assume now that f is irreducible over $\mathbb{F}_q$ and not absolutely irreducible. Then Kaltofen's algorithm can be used to find a field extension K of $\mathbb{F}_q$ with $[K:\mathbb{F}_q] \leq n$ and a proper factorization of f over K. If g and h are two distinct factors, then the first coordinate of any common root is a root of

$$\operatorname{res}_y(g,h) \in K[x].$$

Thus it is easy to calculate all common roots of g and h, to check which ones are in $\mathbb{F}_q^2$, and to determine whether they are indeed roots of f. All roots of f are found in this way; there are at most $n^2/4$ of them (von zur Gathen, Karpinski, and Shparlinski (1996)).

THEOREM 5.1. *Let $f \in \mathbb{F}_q[x,y]$ have degree n. There is a probabilistic algorithm using $(n \log q)^{O(1)}$ operations in $\mathbb{F}_q$ that determines whether f is exceptional and, if it is, finds all points of $\{f = 0\}$.*

REFERENCES

L. M. ADLEMAN AND M.-D. HUANG (2001), *Counting points on curves and abelian varieties over finite fields*, J. Symbolic Comput., 32, pp. 171–189.

D. G. CANTOR AND H. ZASSENHAUS (1981), *A new algorithm for factoring polynomials over finite fields*, Math. Comp., 36, pp. 587–592.

J. VON ZUR GATHEN (1984), *Hensel and Newton methods in valuation rings*, Math. Comp., 42, pp. 637–661.

J. VON ZUR GATHEN AND E. KALTOFEN (1985), *Factorization of multivariate polynomials over finite fields*, Math. Comp., 45, pp. 251–261.

J. VON ZUR GATHEN, M. KARPINSKI, AND I. E. SHPARLINSKI (1996), *Counting curves and their projections*, Comput. Complexity, 6, pp. 64–99.

J. VON ZUR GATHEN AND V. SHOUP (1992), *Computing Frobenius maps and factoring polynomials*, Comput. Complexity, 2, pp. 187–224.

J. VON ZUR GATHEN AND I. E. SHPARLINSKI (1998), *Computing components and projections of curves over finite fields*, SIAM J. Comput., 28, pp. 822–840.

M.-D. HUANG AND D. IERARDI (1998), *Counting points on curves over finite fields*, J. Symbolic Comput., 25, pp. 1–21.

M.-D. HUANG AND Y.-C. WONG (1999), *Solvability of systems of polynomial congruences modulo a large prime*, Comput. Complexity, 8, pp. 227–257.

M.-D. HUANG AND Y.-C. WONG (1998), *An algorithm for approximate counting of points on algebraic sets over finite fields*, in Algorithmic Number Theory, Proceedings ANTS-III (Portland OR), J. P. Buhler, ed., Lecture Notes in Comput. Sci. 1423, Springer-Verlag, New York, pp. 514–527.

E. Kaltofen (1985), *Fast parallel absolute irreducibility testing*, J. Symbolic Comput., 1, pp. 57–67.
R. M. Karp, M. Luby, and N. Madras (1989), *Monte-Carlo approximation algorithms for enumeration problems*, J. Algorithms, 10, pp. 429–448.
A. K. Lenstra (1985), *Factoring multivariate polynomials over finite fields*, J. Comput. System Sci., 30, pp. 235–248.
G. I. Perel'muter (1969), Оценка суммы вдоль алгебраической кривой (*Bounds on sums along algebraic curves*), Mat. Zametki, 5, pp. 373–380.
I. R. Shafarevich (1974), *Basic Algebraic Geometry*, Grundlehren Math. Wiss. 213, Springer-Verlag, New York, Heidelberg.
V. Shoup (1990), *On the deterministic complexity of factoring polynomials over finite fields*, Inform. Process. Lett., 33, pp. 261–267.
I. E. Shparlinski (1999), *Finite Fields: Theory and Computation*, Math. Appl. 477, Kluwer Academic Publishers, Dordrecht, The Netherlands.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1449–1474

ON FRICTIONAL MECHANICAL SYSTEMS AND THEIR COMPUTATIONAL POWER*

JOHN H. REIF† AND ZHENG SUN‡

Abstract. In this paper we define a class of mechanical systems consisting of rigid objects (defined by linear or quadratic surface patches) connected by frictional contact linkages between surfaces. (This class of mechanisms is similar to the analytical engine developed by Babbage in the 1800s, except that we assume frictional surfaces instead of toothed gears.) We prove that a universal Turing Machine (TM) can be simulated by a (universal) frictional mechanical system in this class consisting of a constant number of parts. Our universal frictional mechanical system has the property that it can reach a distinguished final configuration through a sequence of legal movements if and only if the universal TM accepts the input string encoded by its initial configuration. There are two implications from this result. First, the robotic mover's problem is undecidable when there are frictional linkages. Second, a mechanical computer can be constructed that has the computational power of any conventional electronic computer and yet has only a constant number of mechanical parts.

Previous constructions for mechanical computing devices (such as Babbage's analytical engine) either provided no general construction for finite state control or the control was provided by electronic devices (as was common in electromechanical computers such as Mark I subsequent to Turing's result). Our result seems to be the first to provide a general proof of the simulation of a universal TM via a purely mechanical mechanism.

In addition, we discuss the universal frictional mechanical system in the context of an error model that allows an error up to ϵ in each mechanical operation. We first show that, for a universal TM M, a frictional mechanical system in this ϵ-error model can be constructed such that, given any space bound S, the system can simulate the computation of M on any input string ω if M decides ω in space bound S, provided that $\epsilon < 2^{-cS}$ for some constant c. We also show that, for any universal TM M and space bound S, there exists a frictional mechanical system in the ϵ-error model with $\epsilon = \Omega(1)$; it has $O(S)$ parts and can simulate M on any input ω that M decides in space bound S.

Key words. robotics, motion planning, complexity, mechanical computing device

AMS subject classifications. 68T40, 93C85, 03D15, 68Q15, 68Q17

DOI. 10.1137/S0097539798346652

1. Introduction.

1.1. Motivation.

There are two contexts where our result may be of interest: robotic mover's problems and mechanical computing machines.

1.1.1. Robotic mover's problems.

The objective of *robotic mover's problems* (or *motion planning problems*) is to plan the motion of a robot between distinguished configurations under specified physical constraints (e.g., avoiding obstacles) and possibly also dynamic constraints. There are two categories of problems studied in this area: decision problems and optimization problems. The goal of a decision problem is to decide, given the physical and dynamic constraints, whether or not there

*Received by the editors October 29, 1998; accepted for publication (in revised form) June 28, 2003; published electronically September 17, 2003. A preliminary version of this paper appeared in *Proceedings of the 3rd Workshop on Algorithmic Foundations of Robotics* (WAFR98), Houston, TX, 1998. This research was supported in part by grants DARPA/AFSOR F30602-01-2-0561, NSF ITR EIA-0086015, NSF EIA-0218376, and NSF EIA-0218359.

http://www.siam.org/journals/sicomp/32-6/34665.html

†Department of Computer Science, Duke University, Box 90129, Durham, NC 27708-0129 (reif@cs.duke.edu).

‡Department of Computer Science, Hong Kong Baptist University, Kowloon, Hong Kong (sunz@comp.hkbu.edu.hk).

is a sequence of legal movements that will allow the robot to reach the goal configuration; the goal of an optimization problem is to find an optimal path (trajectory) that leads to the goal configuration, according to a predefined cost function on paths (trajectories).

The first hardness result for a robotic mover's problem was presented by Reif [9]. He showed that the generalized mover's problem of moving n linked polyhedra through a set of three-dimensional (3D) obstacles is PSPACE-hard. His proof used a reduction of the computation of any reversible Turing machine (TM) on an input string to an instance of the mover's problem. Hopcroft, Joseph, and Whitesides [7] improved on this result by proving that the mover's problem for two-dimensional (2D) linkages is PSPACE-hard. Later, the generalized mover's problem was proved to be in PSPACE by Canny [2]. Using a *path coding* technique, Canny and Reif [3] also proved that computing the shortest path for a point robot moving amidst polyhedral obstacles is NP-hard. Asano, Kirkpatrick, and Yap [1] introduced the problem of computing the d_1-optimal motion for a 2D rod (defined by a directed line segment) amidst polygonal obstacles and showed that this problem is NP-hard.

There are also many hardness results on various extensions of the basic robotic mover's problem, such as moving obstacles, multirobots, etc. For example, in [8] Hopcroft, Schwartz, and Sharir proved that motion planning for multiple independent rectangular boxes sliding inside a rectangular box is PSPACE-hard. A similar problem of moving multiple discs inside a polygon in a 2D space, however, could be proved only to be strongly NP-hard [14]. Reif and Sharir [10] introduced the 3D mover's problem in the presence of moving obstacles and showed that this problem is PSPACE-hard even in a case where the object to be moved is a disc with bounded velocity. By extending the path coding technique of [3], Reif and Wang [12] proved that the 2D curvature-constrained shortest-path problem is NP-hard. Later, Reif and Sun [11] showed that the time-optimum path planning problem for a point robot in a 3D space composed of polyhedral regions with flows is PSPACE-hard.

Most robotic mover's problems assume that the obstacles are the only objects in the robot's workspace besides the robot itself. One exception is the *movable object problem*, which is to ask whether a robot can move certain objects amidst obstacles in a space to reach a target configuration. That is, there are two types of objects in the space: obstacles that cannot be moved and penetrated (or even touched) by the robot, and objects whose placements can be changed by the robot. The goal of the robot is either to rearrange the movable objects in the space to a desired configuration or reach a target configuration of itself, or both. The first result on the movable object problem was given by Wilfong [15], who studied this problem for the case of a polygonal robot moving in translation amidst polygonal movable objects in a bounded polygonal space. He proved that if the final configurations of the objects are not specified as part of the goal of the motion planning problem, this problem is NP-hard; otherwise, it is PSPACE-hard. He also gave one algorithm for each of the two cases where only one movable object is present.

In Wilfong's model, a robot can grasp an object only from a finite number of positions. This problem is considered to be more difficult when this number is infinite. Chen and Hwang [4] gave a heuristic algorithm to solve one model of this problem where the total weight of objects moved by a robot is to be minimized. Dacre-Wright, Laumond, and Alami [5] extended Wilfong's work by providing an $O(n^3 \log n)$ algorithm for the infinite grasping position case where the final configurations of the objects are specified as part of the goal.

1.1.2. Mechanical computing machines. In 1822, Charles Babbage designed and constructed the difference engine. This machine was specially designed for the evaluation of polynomials. Later, in 1833, Babbage proposed (but did not fully construct) a new device, the analytical engine, which was conceived to solve general arithmetic problems. It resembled the modern digital computer in the following ways.

Input device. Just like any modern computer, the analytical engine was designed to have an input mechanism, such as punch cards.

Memory. The analytical engine was supposed to store the data encoded by mechanical positions; e.g., distinct digits were stored via rotational positions of distinct mechanical dials.

Arithmetic unit. The machine was able to manipulate the data and, in particular, to execute the arithmetic operations. This was done by a part called the "mill" using various gearing mechanisms.

Control unit. The machine was also envisioned by Babbage to have a mechanism that could control the sequence of operations to carry out the computations.

About half a century after Babbage's death, Dr. Vannevar Bush resumed the work of building a mechanical computing device. In 1925 he, along with some associates, made a mechanical calculator powered by an electric motor. His machine was an analog one, in the sense that arithmetic operations were carried out by mechanical means and in terms of physical measurements.

In 1939, Howard Aiken, in collaboration with four IBM engineers, built a general purpose computing machine, the so called automatic sequence controlled calculator, Mark I. Just like Babbage's analytical engine, Mark I performed computation by manipulating mechanical devices. The key difference between Mark I and the previous mechanical computers such as the analytical engine and Bush's machine is that, while the analytical engine and Bush's computer were purely mechanical, the operations of mechanical parts of Mark I were controlled electrically.

As electronic devices were not available in the 1800s, Babbage had to exploit a purely mechanical system to build a computer. Subsequent electromechanical computers could exploit electronics for control, and of course so do the modern computers. Today, Babbage's concept of a purely mechanical computer would at first seem to be out of date, as computers built by much faster electronic technology prevail in every corner of the world. However, the emergence of nanotechnology provides new motivation on studies of mechanical computers.

1.2. Our results.

1.2.1. Frictional mechanical system and frictional mover's problem. All the robotic mover's problems mentioned above assume that there is no friction between objects, and most of the models allow only collision-free movements so that different objects cannot even make contact with each other. The only work that addressed motion planning in the presence of friction is by Sellen [13]. He proved that the dynamic motion planning problem with forbidden movements (in particular, sliding) is undecidable by showing that the actions of a TM can be realized by logical and arithmetic operations, which can be implemented by mechanical means. However, in his model, the motions of the objects corresponding to the computations of the TM cannot be generated deterministically. Therefore, this model cannot be used for constructing a mechanical computer.

We define a *frictional mechanical system* to be a collection of rigid objects in 3D space whose surfaces are composed of linear or quadratic surface patches specified by rational coefficients. All objects are nonpenetrable; i.e., the only allowed intersection

is via surface contact. Each surface patch of each object is also specified as either *frictional* or *sliding* (nonfrictional). If two objects with frictional surfaces make contact with each other, it is assumed that there is no sliding[1] between them. If at least one of the two contacting surfaces is designated to be a sliding surface, there will be no friction between them so that they can slide freely. Furthermore, some objects in the space are specified to move monotonically. In particular, there may be some discs in the system that can rotate only in one direction (clockwise or counterclockwise). We say that the objects can be moved *legally* if all the above constraints are satisfied.

We define the *resource bound*, denoted by R, to be the number of distinct objects in the frictional mechanical system. As each object can be specified by a constant number of surface patches, each of which can be specified by a constant number of rational coefficients, the total number of binary bits used to encode an object is bounded by a constant. Hence, the total number of bits in the binary representation of the frictional mechanical system is $O(R)$.

The *frictional mover's problem* is to determine whether these objects in the frictional mechanical system can be moved legally from a specified initial configuration to a specified final configuration. This problem can be regarded as a generalization of the movable object problem. Compared to the previous works, our model is in 3D space, and the surfaces of the objects in the space can be nonlinear. Further, in addition to moving an object by grasping or pushing (directly or indirectly), a robot in our model can move objects by using the friction between it and surrounding objects. More specifically, a *power disc* in the frictional mechanical system can be deemed as a rather "dumb" robot; it is restricted to rotate in a specified direction without translation. And the problem is to ask whether this robot can rearrange the objects in the system to a target configuration by its rotation.

We prove that the frictional mover's problem is undecidable by reducing the acceptance problem for a Turing machine[2] (TM), A_{TM},[3] to the frictional mover's problem. Given a universal TM[4] M, we construct a frictional mechanical system to simulate this machine. This frictional mechanical system will have the property that the objects in this system can be moved from an initial configuration, which encodes an input string ω of M, to a configuration corresponding to the accepting state of M if and only if M accepts ω. Therefore, as the acceptance problem for a TM is undecidable, so is the frictional mover's problem. This implies that there is no realistic machine that can solve this problem.

An interesting property of this frictional mechanical system is that if M accepts ω, there will be a unique simple path (i.e., one that does not repeat the same configuration) from the initial configuration to the final configuration.

The proof will actually construct, for any given TM M, a frictional mechanical system that simulates M. Every movable object in the system is engaged or linked directly or indirectly with the power disc so that, when the power disc rotates, it will make those objects move accordingly. For any input string ω of M, this frictional

[1]Sliding is a move in a direction tangent to the surfaces at the contact point.

[2]A Turing machine is an abstract machine with a finite state control and a tape that can store an infinite string of symbols. There is a read-write head on the tape that allows the machine to read or write the symbol at the current position of the head. The machine can write to the current position and move the head left or right according to the current state and the current symbol in a specified way.

[3]The acceptance problem A_{TM} for a TM is to determine, given the description of a TM M and its input ω, whether M accepts ω.

[4]A universal TM M will take the description of any TM M' and any input string ω of M' as an input and simulate the behavior of M' presented with input ω.

mechanical system can be set to an initial configuration encoding ω so that, after the power disc has rotated a sufficient number of cycles, this system will result in a configuration encoding the accepting state (rejecting state) of M if and only if M accepts (rejects) ω.

Even though Babbage claimed that the analytical machine could be used to solve any arithmetic problem, it is doubtful whether this machine is as powerful as a universal computing machine. The difficulty is that Babbage did not have a general concept of an abstract computing machine, such as a TM, which was later introduced by Alan Turing in 1936. To show that a mechanical system has the computational power of an electronic computer, it is sufficient and necessary to show that this mechanical system can simulate a universal TM, as proposed by Turing. Although Babbage designed his analytical machine to have a control unit used to guide the arithmetic operations, it was not explicitly shown by Babbage how it could simulate a general finite state control, which is the core of a universal TM. The electromechanical computer, Mark I, was designed to be capable of fulfilling any computing tasks and thus should be as powerful as the modern computers. However, it used an electronic device as the central controller. To our knowledge, our mechanical system, as it will be described in later sections, is the first mechanical system that can perform general-purpose computation without using any electronic devices.

Another limitation of Babbage's analytical engine is its representation of numbers. The analytical engine was a digital computer. Each digit of a number was represented by a mechanical device, such as a dial. The accuracy of Babbage's machine, in terms of the number of decimal places, was determined by the number of dials used to represent a number. Therefore, no matter how precisely the machine was built, it would not improve its accuracy. The only way to improve the accuracy of the machine's computation (or increase the number of data bits) is to increase the number of dials used to represent a number. This will change the structure of the analytical engine and make it bigger and more complicated. Our frictional mechanical system, however, is an analog computer, in the sense that a number (in fact, the entire data of the computation) is represented by the rotational position of a single disc. The accuracy of our system depends on the accuracy of the measurement and the precision of the mechanical devices. Therefore, it can be arbitrarily accurate as we reduce the error in transitional (or rotational) measurement as well as the error in building mechanical devices.

For our proof of the undecidability of the frictional mover's problem, we adopt a simple deterministic model for frictional contacts between objects. Note that there exists a number of considerably more complex models (for example, see [6]) for frictional contacts between objects where the objects in contact with surfaces may make nondeterministic motions. However, our simple deterministic model for frictional contacts will suffice for us to adequately model frictional contacts in the simple cases we employ in our constructions and thus to prove our undecidability result for movement planning with frictional contacts. Moreover, many of these more complex models for frictional contacts reduce to our simple deterministic model for frictional contacts in the simple cases we employ in our constructions.

1.2.2. Frictional mechanical system with error. We prove that a frictional mechanical system can be constructed to simulate a universal TM. Therefore, this system can be used for arbitrary finite computation, just like any conventional computer. However, the underlying assumption is that this frictional mechanical system can be constructed exactly as it is specified. We are also interested in the computa-

tional power of such a frictional mechanical system in the case where inaccuracy is allowed in the construction of the mechanical devices in the system.

There are many factors that might induce errors in the computations of a frictional mechanical system, including the precision of manufacturing the parts. For example, the circumference of a disc may not exactly be manufactured to be a circle. The radius of a disc may not be manufactured to be exactly as it is specified. When two discs are very close but still not in contact with each other theoretically, they may already have surface contact so that the rotation of one disc will move the other one, even though they are not supposed to do so.

Since there are a constant number of mechanical devices in our mechanical system, we can let ϵ be the upper bound for the errors that occur in a single operation. This is our ϵ-error model. We prove that, given a space bound S, our frictional mechanic system in this ϵ-model can simulate the universal TM M on any input string ω that can be decided by M in space bound S, provided that $\epsilon = O(2^{-cS})$ for some constant c.

We also prove that, given a universal TM M with space bound S, there exists an $\epsilon = \Omega(1)$ such that a frictional mechanical system in ϵ-error model can simulate the computation of M presented with any input ω if M decides ω in space bound S. This result provides decreased required precision of parts at the expense of increased number of parts, which increases with S.

1.3. Notation. In the following sections, the universal TM with end-marks is denoted by quintuple $M = (Q, \Sigma, \delta, q_0, \{q_{\sigma-2}, q_{\sigma-1}\})$ as follows:

1. $Q = \{q_0, q_1, \ldots, q_{\sigma-1}\}$ is the set of states.
2. $\Sigma = \{0, 1, \ldots, m-1\}$ is the tape alphabet. Here 0 denotes the *blank* symbol. 1 and 2 are the left and right end-marks, respectively.
3. $\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$ is the transition function.
4. q_0 is the start state.
5. $\{q_{\sigma-2}, q_{\sigma-1}\}$ is the set of halting states. In particular, $q_{\sigma-2}$ is the rejecting state and $q_{\sigma-1}$ is the accepting state.

In our following discussion, m and σ are considered as constants.

At any time during the computation, we use the *current working space* of the TM to denote the portion of the tape that the read-write head has visited so far. The current working space is always bounded by a left end-mark and a right end-mark. We denote the tape status by a string $\omega_1\omega_2\cdots\omega_{k_1-1}\check{\omega_{k_1}}\omega_{k_1+1}\cdots\omega_{k_2}$. Here ω_1 and ω_{k_2} are the left and right end-mark, respectively, and $\check{\omega_{k_1}}$ denotes that the read-write head is at the k_1th cell of the tape. The read-write head will never replace the left end-mark by another symbol nor will it go beyond the left end-mark. Whenever the read-write head replaces the right end-mark by another symbol, it will pad a right end-mark to the right of the symbol.

It is easy to see that, for any TM (without end-marks) M', there is an equivalent TM with end-marks as described above. Therefore, all universal TMs mentioned in the following discussion are assumed to have end-marks.

Since discs, in particular the rotational positions of discs, play a very important role in our frictional mechanical system, we want to specify several terminologies which we will be using frequently in the following discussion.

Each disc used in our system has a specified orientation called the *initial orientation*. Therefore, the rotational position of a disc D can be specified by the angle the disc has rotated from its initial orientation. For our convenience, we will use "the angle of D" to denote this angle. Also, we will say that the angle recorded (or

represented) by the rotational position of disc D is θ. Further, if we say "to increase (decrease) the angle or rotational position of a disc D by an angle of θ," we mean to rotate D counterclockwise (clockwise, respectively) by an angle of θ.

A *partial disc* is a portion of a disc bounded by two radii and the remaining portion of the circumference. As we will show later, partial discs are very useful in our system too.

1.4. Organization of this paper. Section 2 presents several basic mechanical devices widely used in our frictional mechanical system. After that, we provide the full description of the frictional mechanical system that simulates a universal TM. In the last section, the model of the frictional mechanical system with errors is discussed.

2. Basic gadgets. We prove the reduction from A_{TM} to the frictional mover's problem by constructing a frictional mechanical system for a universal TM M. This system is composed of discs, partial discs, cylinders, and other geometric objects. There is a special disc, the power disc, which is specified to rotate only clockwise. The rotation of this disc will force other objects in the system to move, due to frictional linkages between them. This frictional mechanical system can "simulate" M in the sense that, for any input string ω of M, this system can be set to a corresponding initial configuration so that a distinguished final configuration can be reached if and only if M accepts ω. Therefore, if one can decide this instance of the frictional mover's problem, he can also decide the corresponding instance of A_{TM}. As it is well known that A_{TM} is undecidable, so must be the frictional mover's problem.

Friction is very important in this system, as it not only provides a method of moving other objects in the system but also preserves the state of the system and thus the state of the TM the system simulates. It also guarantees the system properly transfers from one state to another.

Before giving the construction of the frictional mechanical system, we first introduce in this section several "gadgets" that perform the basic functionalities and thus are used widely in our system.

There are seven kinds of basic gadgets:

Converting device. A converting device is used to convert between rotational displacement and transitional displacement.

Sequencer. A sequencer is composed of a group of discs and partial discs that share the same axis and rotate with the same angular speed at any time. These discs (and partial discs) are engaged with some surrounding discs in the frictional mechanical system so that, when rotating, they will make the surrounding discs rotate with them and in turn move other objects in the system. The sequencer is a key component in the frictional mechanical system, as it provides a mechanism for moving all the objects in the system in a specified way.

Transitional movement sequence controller. A transitional event sequencer controller allows the sequencer to periodically move an object in the system by a constant distance and then move it back.

Rotational movement sequence controller. A rotational event sequencer controller allows the sequencer to periodically rotate a disc by a constant angle.

Resettable rotational disc. A resettable rotational disc has two states: the free rotation state, when it can be rotated by engaging with another disc, and the reset state when it will return to a specified initial orientation no matter how much it is rotated in the free rotation period. Virtually all devices need to use resettable rotational discs.

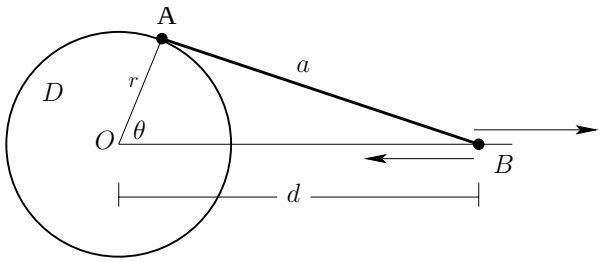


FIG. 1. *Converting device.*

Nonlinear mapping controller. A nonlinear mapping device is used to implement a nonlinear mapping between rotations. This is a very crucial component in the construction of the simulation of a finite state control.

Selection controller. A selection controller helps the sequencer to initiate one among several event sequences, depending on the rotational (or transitional) position of an object in the system. This is yet another key component in the construction of the simulation of a finite state control.

In the following subsections, we will give a detailed description for each of the basic gadgets.

2.1. Converting device. The most basic device is one that can convert a rotational displacement to a transitional displacement or vice versa. This can be done by the device described in Figure 1.

As can be seen from the figure, one end A of an arm of length a is attached to a fixed position on the rim of the disc D. The other end B of this arm is restricted to move along a line l passing the center of the disc. The rotation of the disc D, θ, and the distance d between the center of the disc and B satisfy $d = r\cos\theta + \sqrt{a^2 - r^2\sin^2\theta}$. Here r is the radius of D.

Therefore, if θ varies between $\pi/4$ and $3\pi/4$, then d will vary between $\sqrt{a^2 - r^2/2} - r/\sqrt{2}$ and $\sqrt{a^2 - r^2/2} + r/\sqrt{2}$. Further, d is a monotone function of θ. Therefore, the rotational displacement of disc D can be converted to the transitional displacement of B and vice versa.

2.2. Sequencer. All the objects in the system are moved, directly or indirectly, by a mechanism called a sequencer, which is composed of a group of discs and partial discs. These discs and partial discs share the same axis and rotate with the same angular speed at any time. Each disc or partial disc is engaged with one or more surrounding discs. Therefore, when the sequencer is rotating, it will make these surrounding discs rotate. Also, the rotation of these surrounding discs will then move other objects in the system.

In Figure 2, there are three discs and two partial discs in the sequencer. Each of the discs of the sequencer will make its surrounding disc rotate with it, possibly at a different angular speed. Each of the partial discs, however, will move the surrounding disc for a certain period and then lose contact with it.

Each cycle of the sequencer finishes one step of computation of the universal TM it simulates. As the rotation of the sequencer determines the starting and finishing of any movement sequence of any other object linked with it, we may use an interval $[\theta_1, \theta_2]$ to denote the time period during which the sequencer rotates from angle θ_1 to angle θ_2 in each cycle rather than using the actual time.

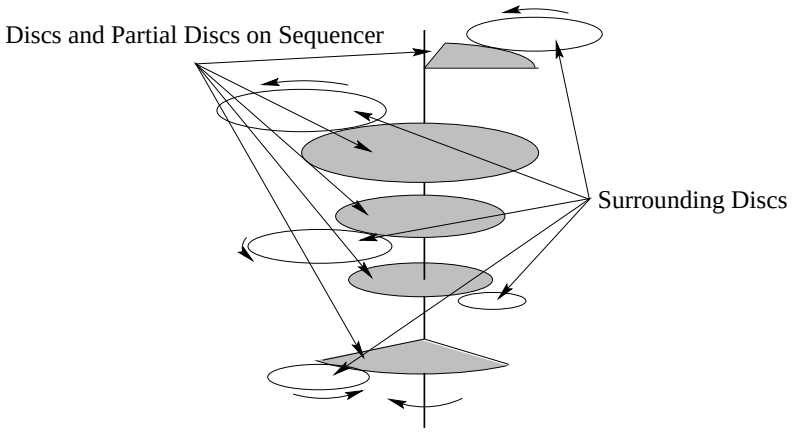


FIG. 2. *Sequencer.*

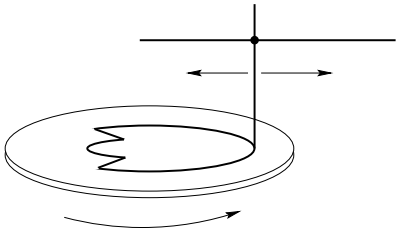


FIG. 3. *Transitional movement sequence controller.*

2.3. Transitional movement sequence controller. It will be useful if the sequencer can move an object to a specified position during time period $[\theta_1, \theta_2]$ of each rotational cycle and move it away from that position otherwise. This can be done by the controller shown in Figure 3.

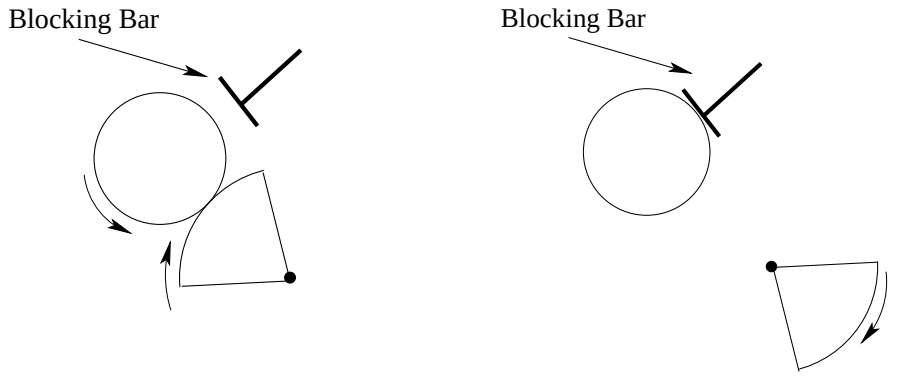
There is a circular gap on the surface of the disc which is on the sequencer. The curve of this gap is defined by $r = r(\theta)$ in a polar coordinate with origin at the center of the disc. A vertical bar is restricted in the circular gap. Also, it is restricted to move on a line passing the center of the disc. $r(\theta)$ has the property that

$$\begin{aligned} r(\theta) &= r_0 \quad \text{for } \theta \in [\theta_1, \theta_2], \\ r(\theta) &> r_0 \quad \text{for } \theta \notin [\theta_1, \theta_2]. \end{aligned}$$

Here r_0 is a constant. Therefore, when the controller is rotating in one direction, the bar will move back and forth along the line. More specifically, in time interval $[\theta_1, \theta_2]$, the vertical bar is at the position of $(r_0, 0)$ (in a polar coordinate). At any other time, it is at some position $(r', 0)$, where $r' > r_0$.

For our convenience, in the following discussion, we may say "the sequencer moves an object to a certain position during a certain time period in each rotational cycle," implicitly assuming that we have constructed a transitional movement sequence controller, as described here, that fulfills this task.

Transitional movement sequence controllers will be used frequently in our frictional system. All the rotational discs, except those on the sequencer, need to be clamped down whenever they are not rotated, directly or indirectly, by the sequencer so that their orientations will be maintained. This can be done by moving a *blocking*



(a) Rotating period (b) Clamping period

FIG. 4. *Rotational movement sequence controller.*

bar (a bar with a frictional surface) to a position where it has surface contact with the disc in the clamping period of each cycle.

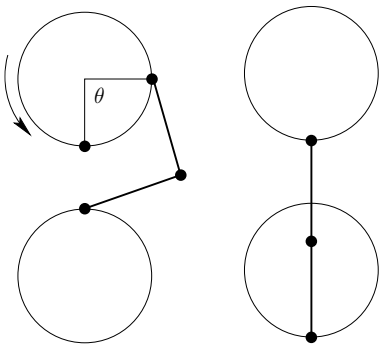
2.4. Rotational movement sequence controller. A rotational movement sequence controller is a partial disc in the sequencer used to rotate another disc, called a *target disc*, in a specified time interval during each cycle (see Figure 4). With a transitional movement sequence controller, a blocking bar can be moved to clamp the target disc when the partial disc loses contact with the target disc. When the partial disc rotates to such an orientation that its rim contacts the rim of the target disc, this bar is moved away from the target disc so that this disc can be rotated by the partial disc.

Therefore, the disc will stay stationary during the clamping period and rotate with the partial disc otherwise. The length of the rotating period is determined by the angle of the partial disc. And the total angle the target disc rotates during the rotating period is determined by the length of the arc of the partial disc as well as the radius of the target disc. Therefore, by setting these parameters appropriately, we can let the target disc rotate by a certain angle, say, π in each cycle.

Again, for our convenience, in the following discussion, we will say "the sequencer rotates a disc by a specified angle during a certain time period," implicitly assuming that we have constructed a rotational movement sequence controller, as described in this subsection, to do it.

2.5. Resettable rotational disc. With a rotational movement sequence controller, the sequencer can rotate a disc by a constant angle during each cycle. Frequently, it is necessary to rotate a target disc by a certain angle θ recorded by a *source disc* and then reset it by rotating it by θ in the reverse direction, no matter what θ is. This kind of disc is called a *resettable rotational disc*. As shown in Figure 5, two discs are linked by a two-segment arm. Each of these two discs has a radius of 1. The distance between the centers of the two discs is d, and the total length of the two-segment arm is d too.

The upper disc is the target disc. The lower disc connects to a rotational movement sequence controller so that at the time when we want to reset the target disc,



(a) Free rotating period (b) Resetting period

FIG. 5. *Resettable rotational disc.*

the lower disc will rotate by an angle of π. Because the length of the two-segment is exactly the same as the distance between the centers of two discs, the upper disc will be forced to rotate to such an orientation that the point where the arm is attached is at the lowest position. When it is time to release the target disc and allow it to be rotated freely by another disc it engages with, the lower disc rotates by another π so that it restores its initial orientation.

Therefore, each resettable rotational disc can be regarded as a register. It can hold a value that is represented by its current angle θ.

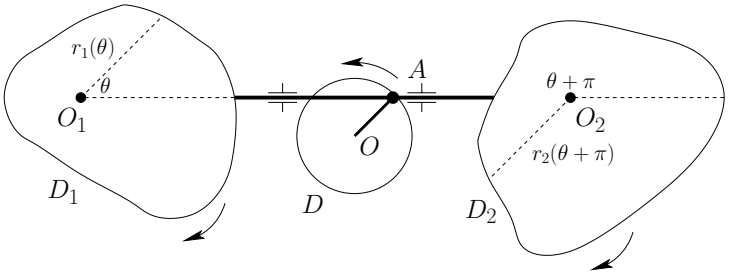
Using resettable rotational discs, we can construct a mechanical device that will increase the angle of a disc by the angle recorded by another disc. Suppose we have two resettable rotational discs. Disc D_1 is set at θ_1, and D_2 is set at θ_2. We want to increase the angle of D_2 by θ_1 while having the angle of D_1 remain unchanged. A third disc D_3 is needed here, and it is set at its initial orientation. The radii of D_1, D_2, and D_3 are the same.

First, both D_2 and D_3 are engaged with D_1, and then D_1 is reset to its initial orientation. D_2 (and D_3) will be rotated counterclockwise by an angle of θ_1. This will set D_2 to angle $\theta_1 + \theta_2$ and D_3 to angle θ_1. Then D_2 is disengaged from D_1. The next step is to reset D_3 to its initial orientation. As D_3 is still engaged with D_1, D_1 will be rotated counterclockwise by an angle of θ_1. Therefore, the orientations of D_1 and D_3 will be set just the same as they were before, while the angle of D_2 is increased by θ_1, which is the angle recorded by D_1.

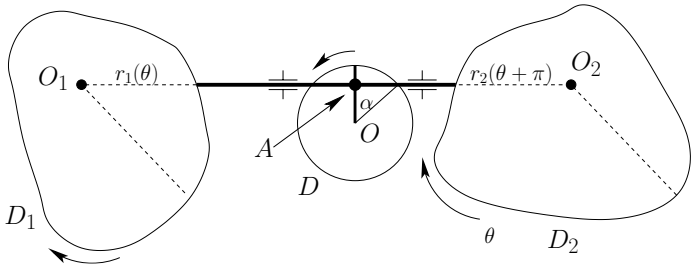
Observe that we can amplify or reduce θ_1 by a constant factor C before adding it to D_2. This can be done by setting $r_1 = r_3 = Cr_2$, where r_1, r_2, and r_3 are the radii of the three discs, respectively.

Here we will digress to address the necessity of using discs with frictional linkages. If gears instead of discs were used, it might be possible that teeth of the gears could not match after they resume contact with each other, since the two gears might rotate by a different angle after they are disengaged.

2.6. A nonlinear mapping device. Engaging a source disc and a target disc will allow us to perform linear mapping from the angle of rotation of the source disc to the angle of rotation of the target disc. By "linear mapping" we mean that if the



(a) Initial position



(b) After rotating by a degree of θ

FIG. 6. *Nonlinear mapping device.*

source disc rotates by an angle of θ, the target disc will rotate by an angle of $C \cdot \theta$, where C is the radius ratio between the two discs. However, sometimes it is necessary to perform nonlinear (or even nonmonotone) mapping on the rotations of discs. In particular, a device with the following property is needed.

PROPERTY 1. *A nonlinear mapping device has two discs, an input disc and an output disc, each of which can rotate by any angle* $\theta \in [0, \pi/2]$ *from its initial orientation. Given* $\theta_1, \theta_2, \ldots, \theta_n$, $\alpha_1, \alpha_2, \ldots, \alpha_n$, $0 \le \theta_1 < \theta_2 < \cdots < \theta_n \le \pi/2$, $0 \le \alpha_i \le \pi/2$, *if the rotation of the input disc is* θ_i, *the rotation of the output disc should be* α_i. *Further, if* $\alpha_1 < \alpha_2 < \cdots < \alpha_n$, *the mapping should be monotone. That is, for any* θ, $\theta_i < \theta < \theta_{i+1}$, *if the rotation of the input disc is* θ, *the rotation of the output disc,* α, *should be in the range of* (α_i, α_{i+1}).

Nonlinear mapping can be implemented by the device described in Figure 6. It consists of four parts, a central (regular) disc D centered at point O, a horizontal bar, and two irregular discs D_1 and D_2 centered at O_1 and O_2, respectively. In each operation D_1 and D_2 will rotate clockwise by an angle of θ_i for some $i, 1 \le i \le n$. The motion of D_1 and D_2 will cause D, through the horizontal bar connecting them, to rotate by an angle of α_i from its initial orientation.

The horizontal bar, which lies on line $\overline{O_1O_2}$ between D_1 and D_2, is restricted to move horizontally only. Each end of the horizontal bar touches, but is not attached to, the rim (circumference) of one of the two irregular discs. The bar is connected to D through a joint A between the bar and a spoke (a radiating bar from the center of a disc to its circumference) of D. The joint has the property that it can move along

the spoke but will remain stationary with respect to the bar. Therefore, when D_1 and D_2 rotate, they will in turn push the bar back and forth horizontally, which in turn will move the spoke and thus make D rotate accordingly.

During each operation, D_1 and D_2 need to "complement" each other in the sense that they keep constant contact with the horizontal bar. Nor can the two irregular discs "squeeze" the bar, as we assume that all objects are rigid and cannot be compressed. Therefore, we want to design the shapes of D_1 and D_2 in such a way that, when D_1 and D_2 rotate clockwise with the same angular speed, the distance between D_1 and D_2 on line $\overline{O_1O_2}$ is a constant. This implies that $r_1(\theta)+r_2(\theta+\pi)$ is a constant for any $\theta \in [0, \pi/2]$, where, for each $j = 1, 2$, $r_j(\theta)$ is the distance between O_j and the intersection point of the rim of D_j and the ray with angle θ that starts from O_j.

Initially, D is set at the orientation such that the angle between the spoke of D and the x-axis is $\pi/4$, as shown in Figure 6(a). We choose the vertical distance between O and the horizontal bar to be $R/\sqrt{2}$, where R is the radius of D. Therefore, the range of the angle between the spoke and the x-axis is $[\pi/4, 3\pi/4]$. Let h_1 be the horizontal distance between O and O_1, and let d_1 be the length of the part of the horizontal bar between its left end and the joint A with the spoke. To achieve the desired property, it is sufficient that the following equality holds:

$$\frac{r_1(\theta_i) + d_1 - h_1}{R/\sqrt{2}} = \tan(\pi/4 - \alpha_i) \text{ for } i = 1, 2, \ldots, n. \tag{1}$$

This can be done by choosing $r_1(\theta_i)$ to be $h_1 - d_1 + R\tan(\pi/4 - \alpha_i)/\sqrt{2}$ for all $i, 1 \le i \le n$. Also, the initial orientation of D implies that $\alpha = 0$ if $\theta = 0$. Therefore, we have $r_1(0) = h_1 - d_1 + R/\sqrt{2}$. For our convenience, we let $\theta_0 = 0$ and $\alpha_0 = 0$.

To ensure that the mapping is monotone in interval $[\theta_i, \theta_{i+1}]$, for any $\theta \in (\theta_i, \theta_{i+1})$, we specify

$$r_1(\theta) = \frac{(\theta - \theta_i) \cdot r_1(\theta_{i+1}) + (\theta_{i+1} - \theta) \cdot r_1(\theta_i)}{\theta_{i+1} - \theta_i}. \tag{2}$$

Note that $r_1(\theta)$ is a linear function of θ inside interval (θ_i, θ_{i+1}). If $\alpha_1 < \alpha_2 < \cdots < \alpha_n$, the nonlinear mapping will be monotone in interval $[0, \theta_n]$.

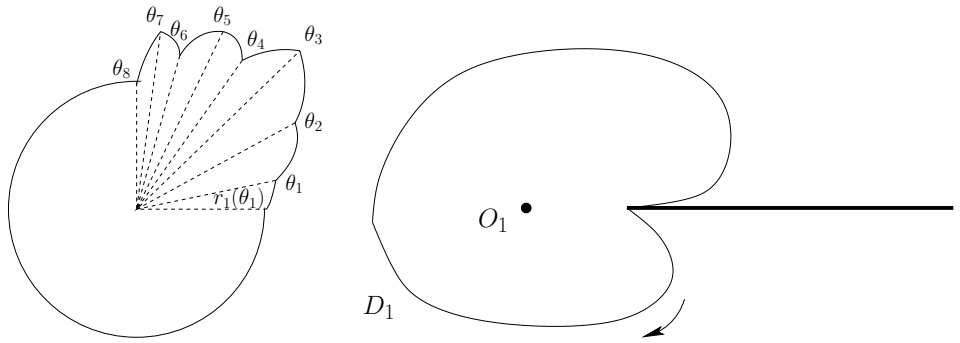
In the above we have shown how to decide $r_1(\theta)$ for any $\theta \in [0, \theta_n]$. For any $\theta \in (\theta_n, 2\pi)$, we let

$$r_1(\theta) = \frac{(\theta - \theta_n) \cdot r_1(0) + (2\pi - \theta) \cdot r_1(\theta_n)}{2\pi - \theta_n}.$$

This completes the specification of D_1. For D_2, we need only to let $r_2(\theta) = |\overline{O_1O_2}| - d - r_1(\theta - \pi)$ so that $r_1(\theta) + r_2(\theta + \pi)$ is a constant for any $\theta \in [0, 2\pi)$. Here d is the length of the horizontal bar.

As shown in Figure 7(a), the boundary of each irregular disc D_j described above is piecewise smooth, with derivative $r_j'(\theta)$ defined everywhere except for $\theta = \theta_0, \theta_1, \ldots, \theta_n$. At these values, function $r_j(\theta)$ has a "left derivative" $r_j'(\theta^-)$ and a "right derivative" $r_j'(\theta^+)$, which in most cases are different. The difference between $r_j'(\theta^-)$ and $r_j'(\theta^+)$ would not hinder the proper functioning of the device.

The only concern is that there should not be a "dimple" on the boundary, where $r_j'(\theta^+) = +\infty$ for some $\theta \in [0, 2\pi)$. In this case, the irregular disc would not be able to push the horizontal bar away from it, and, even worse, the bar would actually prohibit the irregular disc from further rotating, as shown in Figure 7(b). This would lead to a



(a) The boundary is piecewise smooth

(b) A dimple that causes crash

FIG. 7. *Irregular discs.*

complete stop of the computation of the frictional mechanical system. However, since inside each interval $[\theta_i, \theta_{i+1}]$ the curve $r_j(\theta)$ is a linear function of θ (see (2)), $r'_j(\theta^+)$ cannot be $+\infty$ as long as each $r_j(\theta_i)$ is chosen to be a finite number. Therefore, we have ruled out this potential difficulty.

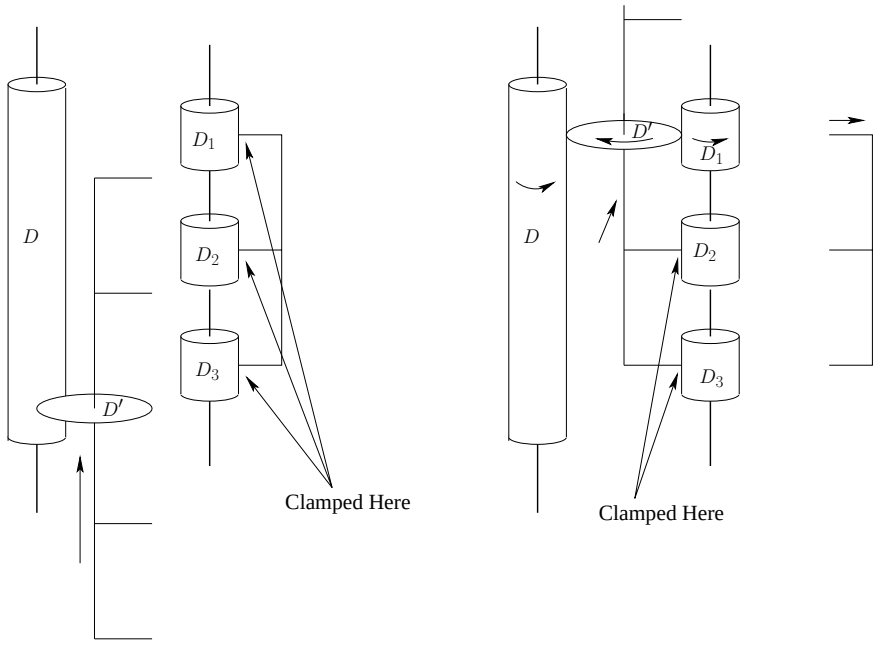
Observe that even though the above mapping device can implement only a mapping from $[0, \pi/2]$ to $[0, \pi/2]$, with a linear mapping device that can amplify or reduce angle, for any nonlinear mapping, we can construct a device to simulate it.

2.7. Selection controller. Another useful device is the so-called *selection controller*, which will rotate one of several cylinders, depending on the transitional displacement of a bar from its initial position or, equivalently, the rotational displacement of a disc from its initial orientation. Therefore, different positions of the bar will incur different movements of the objects in the space.

Suppose there are n cylinders, called *choice cylinders*, from which one is chosen to be rotated, depending on the transitional displacement of a sliding bar. Let l denote the transitional displacement of the sliding bar. We assume that $0 < l < 1$. Further, if $2i/(2n-1) < l < (2i+1)/(2n-1)$ for some i, $0 \le i \le n-1$, the selection controller will select the ith choice cylinder to rotate. This can be implemented by the device in Figure 8.

To the left there is a long cylinder S engaged with a rotational movement sequence controller so that S is rotated by an angle of 2π during a certain period of each cycle and remains static at any other time. To the right there is a pile of n identical cylinders, $D_0, D_1, \ldots, D_{n-1}$. These cylinders share the same axis, but each can rotate independently. The height of each cylinder is $1/(2n-1)$, and the distance between two contiguous cylinders is also $1/(2n-1)$. The radii of $D_0, D_1, \ldots, D_{n-1}$, and D are all r. The distance between D's axis and D_is's axis is $2r + 2R$.

The sliding bar is attached to what we call a *selective engager* shown in the middle of Figure 8. The selective engager consists of a disc D' with radius R and $2n-2$ blocking bars, each of which has length R. $n-1$ of these blocking bars are above D', and the others are below D'. The vertical distance between any two contiguous bars is $2/(2n-1)$, and so are the distance between D' and the bar immediately above D' as well as the distance between D' and the bar immediately below D'.



(a) Disengaging period (b) Engaging period

FIG. 8. *Selection controller.*

In each cycle, initially the selective engager is placed in such a position that D' is at the same height as the lower face of D_1. Also, the selective engager does not contact either D or any D_is. After the sliding bar moves vertically by a distance of l, $2i/(2n-1) < l < (2i+1)/(2n-1)$ for some i, the selective engager will move upwards by a distance of l accordingly so that it will be at the same height as D_i.

Then the selective engager is moved horizontally to the position right between D and D_is. Since the distance between D's axis and D_is's axis is $2r + 2R$, D' will contact both D and D_i. Also, any of $D_0, D_1, \ldots, D_{n-1}$ other than D_i will contact one of those blocking bars so that they are forbidden from rotating.

After removing the blocking bars originally clamping the n cylinders, only n_i can rotate freely. Therefore, when D is rotated by an angle of 2π, it will rotate D', which in turn rotates D_i. Since D and D_i have the same radius, D_i will be rotated by exactly 2π.

The functionality of this device is just like the *switch–case* structure in a C program. In our remaining section, we will say "If the transitional (rotational) displacement of a bar (disc, respectively) is in the range of (a, b), the sequencer will do ...," implicitly assuming that we have constructed a particular selection controller as described here to do this task.

3. Simulation of a universal TM. Given the basic gadgets described above, we are ready to construct a frictional mechanical system that simulates a universal TM. Our frictional mechanical system consists of two components: a *control component* simulating the finite state control and a *tape component* simulating the read-write

tape. All the objects in the system are linked or engaged with a sequencer, which controls the behavior of those objects. The sequencer is engaged with the power disc that is constantly rotating clockwise. Therefore, the sequencer will be rotating counterclockwise constantly, and it will drive all the objects in the system in a specified way as it rotates.

In the control component, there is a disc D_{state} whose rotational position represents the current state of the universal TM. Similarly, in the tape component, there are three discs $D_{current}$, D_{left}, and D_{right} whose rotational positions together represent the status of the read-write tape. (The status of the tape includes the contents of the tape as well as the current position of the read-write head.)

The control component will take the current symbol and the current state of the universal TM, both of which are represented by rotational positions of certain discs, as input and compute the next state, the next move of the head, and the symbol replacing the current symbol, all of which are also represented by rotational positions. The tape component will then modify the rotational positions of $D_{current}$, D_{left}, and D_{right}, depending on the next move and the replacing symbol provided by the control component.

The first subsection presents the methods used to encode the current state of the universal TM and the current status of the read-write tape by rotational positions. The next two subsections describe the construction of the control component and the tape component, respectively. The last subsection shows how these two components work together to simulate a universal TM and, subsequently, how the two main theorems of this paper are proved.

3.1. Encoding the configuration of the universal TM. The configuration of a TM includes the current state, the current tape contents, and the current head location. To be able to simulate a TM, our frictional mechanical system should be able to record the configuration of the TM by rotational positions of discs in the system.

The status of the read-write tape of the universal TM can be encoded by the rotational positions of three discs $D_{current}$, D_{left}, and D_{right}. If $\omega_1\omega_2\cdots\omega_{k_1-1}\check{\omega_{k_1}}$ $\omega_{k_1+1}\cdots\omega_{k_2}$ is the current tape status, the rotation of $D_{current}$, $\beta_{current}$, the rotation of D_{left}, β_{left}, and the rotation of D_{right}, β_{right} are set as follows:

$$\beta_{current}(\omega_1\omega_2\cdots\check{\omega_{k_1}}\cdots\omega_{k_2}) = \frac{2\omega_{k_1}\pi}{2m+1}, \tag{3}$$

$$\beta_{left}(\omega_1\omega_2\cdots\check{\omega_{k_1}}\cdots\omega_{k_2}) = 2\pi\left(\sum_{i=1}^{k_1-1}\frac{\omega_{k_1-i}}{(2m+1)^i}\right), \tag{4}$$

$$\beta_{right}(\omega_1\omega_2\cdots\check{\omega_{k_1}}\cdots\omega_{k_2}) = 2\pi\left(\sum_{i=1}^{k_2-k_1}\frac{\omega_{i+k_1}}{(2m+1)^i}\right). \tag{5}$$

In particular, $\beta_{left} = 0$ if $k_1 = 1$, and $\beta_{right} = 0$ if $k_1 = k_2$.

Therefore, $\beta_{current}$ encodes the current symbol, and D_{left} (D_{right}, respectively) encodes the substring to the left (right, respectively) of the current symbol.

It is easy to see that this encoding function has the following properties: (i) $(2i-1)\pi/(2m+1) < \beta_{current} < (2i+1)\pi/(2m+1)$ if $\omega_{k_1} = i$; (ii) $(2i-1)\pi/(2m+1) < \beta_{left} < (2i+1)\pi/(2m+1)$ if $\omega_{k_1-1} = i$; and (iii) $(2i-1)\pi/(2m+1) < \beta_{right} < (2i+1)\pi/(2m+1)$ if $\omega_{k_1+1} = i$.

Therefore, linking a selection controller with each of $D_{current}$, D_{left}, and D_{right} will allow the frictional mechanical system to decide the current symbol of the tape and the two symbols next to it, which will be used by the control component and the tape component.

During the simulation of the universal TM, if the current state of the universal TM is q_i, D_{state} will be set at a rotational position of $\lambda_i = i\pi/(2\sigma)$. Observe that every rotational position of D_{state} that represents a state is between 0 and $\pi/2$. Again, linking a selection controller with D_{state} will allow the frictional mechanical system to decide the current state of the tape.

3.2. Finite state control component. We let $\delta_1 : Q\times\Sigma \to Q$, $\delta_2 : Q\times\Sigma \to \Sigma$, and $\delta_3 : Q \times \Sigma \to \{L, R\}$ denote the three components of the transition function δ, respectively. More specifically, if the current state is q_j and the current symbol is i, the next state is $\delta_1(q_j, i)$, the symbol to replace the current symbol is $\delta_2(q_j, i)$, and the next move of the head is $\delta_3(q_j, i)$.

The control component consists of $3m$ nonlinear mapping devices. The first m of these nonlinear mapping devices, $M_{1,0}$, $M_{1,1}$, ..., $M_{1,m-1}$, are used to implement δ_1, the next state function. For each i, $0 \le i \le m-1$, $M_{1,i}$ is designed in such a way that if the input disc of $M_{1,i}$ is rotated by an angle of $\lambda_j = j\pi/(2\sigma)$ from its initial orientation, the output disc will be rotated by an angle of $\lambda_{j'} = j'\pi/(2\sigma)$. Here $\delta_1(q_j, i) = q_{j'}$. In other words, if the input disc is rotated by an angle corresponding to the current state of the universal TM, the output disc will be rotated by an angle corresponding to the next state of the TM.

Suppose the current state is q_j and the current symbol is i. As the current symbol can be determined by applying a selection controller to $D_{current}$, the sequencer will be able to choose $M_{1,i}$ from $M_{1,0}, M_{1,1}, \ldots, M_{1,m-1}$ and rotate the input disc of $M_{1,i}$ by an angle of λ_j, which is recorded by D_{state}. Then D_{state} will be set at the angle recorded by the output disc of $M_{1,i}$ and the nonlinear mapping device $M_{1,i}$ will be reset to its initial position.

In the remaining $2m$ nonlinear mapping devices, $M_{2,0}$, $M_{2,1}$, ..., $M_{2,m-1}$ are used to implement δ_2; and $M_{3,0}$, $M_{3,1}$, ..., $M_{3,m-1}$ are used to implement δ_3. For each $i, 0 \le i \le m-1$, $M_{2,i}$ has the property that if the input disc of $M_{2,i}$ is rotated by an angle of $\lambda_j = j\pi/(2\sigma)$ from its initial orientation, the output disc will be rotated by an angle of $2\delta_2(q_j, i)\pi/(2m+1)$ from its initial orientation. Similarly, for each i, $0 \le i \le m-1$, $M_{3,i}$ has the property that if the input disc of $M_{3,i}$ is rotated by an angle of λ_j, the output disc will be rotated by an angle of $\pi/4$ if $\delta_3(q_j, i) = L$ or $\pi/2$ if $\delta_3(q_j, i) = R$.

Therefore, the next move and the replacing symbol will be represented by rotational positions of certain discs. These rotational positions will be used by the tape component to update the rotational positions of $D_{current}$, D_{left}, and D_{right} according to the transition function δ.

3.3. Read-write tape component. As mentioned above, the control component will generate two rotational positions, one representing the next move and the other representing the replacing symbol. Let $\omega_1\omega_2\cdots\omega_{k_1}^{\check{}}\cdots\omega_{k_2}$ be the current tape status and ω'_{k_1} be the replacing symbol.

For our convenience, we use $\beta_{current,t}$ to denote the angle of $D_{current}$ at the beginning of the tth rotational cycle (of the sequencer). $\beta_{left,t}$, $\beta_{right,t}$, and $\beta_{state,t}$ are defined likewise. First, suppose the next move of the head is to go rightward. There are three cases:

1. $k_1 = k_2$; i.e., the head is at the right end-mark:
The status of the tape will be changed to $\omega_1\omega_2\cdots\omega'_{k_1}\omega_{\check{k_2+1}}$. Here $\omega_{k_2+1} = 2$ (indicating the right end-mark) as the head will pad a right end-mark to the right. Therefore, we have the following recursive equations:

$$\beta_{current,t+1} = 2\frac{2\pi}{2m+1}, \tag{6}$$

$$\beta_{right,t+1} = 0, \tag{7}$$

$$\beta_{left,t+1} = \frac{\beta_{left,t}}{2m+1} + \frac{2\pi\omega'_{k_1}}{2m+1}. \tag{8}$$

2. $k_1 = 1$; i.e., the head is at the left end-mark:
As the universal TM will never overwrite the left end-mark, ω_1 will still be 1 (indicating the left end-mark), and the status of the tape will be changed to $\omega_1\check{\omega_2}\omega_3\cdots\omega_t$. Therefore, we have

$$\beta_{current,t+1} = \frac{2\pi\omega_2}{2m+1}, \tag{9}$$

$$\beta_{right,t+1} = \left(\beta_{right,t} - \frac{2\pi\omega_2}{2m+1}\right)\cdot(2m+1), \tag{10}$$

$$\beta_{left,t+1} = \frac{2\pi}{2m+1}. \tag{11}$$

3. $1 < k_1 < k_2$:
The status of the tape will be changed to $\omega_1\omega_2\cdots\omega'_{k_1}\omega_{\check{k_1+1}}\omega_{k_1+2}\cdots\omega_{k_2}$. Thus, correspondingly, the recursive equations are

$$\beta_{current,t+1} = \frac{2\pi\omega_{k_1+1}}{2m+1}, \tag{12}$$

$$\beta_{right,t+1} = \left(\beta_{right,t} - \frac{2\pi\omega_{k_1+1}}{2m+1}\right)\cdot(2m+1), \tag{13}$$

$$\beta_{left,t+1} = \frac{\beta_{left,t}}{2m+1} + \frac{2\pi\omega'_{k_1}}{2m+1}. \tag{14}$$

Similarly, if the next move is to go leftward, there are two cases (the read-write head cannot go leftward if it is at the left end-mark):

1. If $k_1 = k_2$, i.e., the head is at the right end-mark:

$$\beta_{current,t+1} = \frac{2\pi\omega_{k_1-1}}{2m+1}, \tag{15}$$

$$\beta_{right,t+1} = \begin{cases} \frac{2\cdot 2\pi}{2m+1} & \text{if } \omega'_{k_1} = 2, \\ \frac{2\pi\omega'_{k_1}}{2m+1} + \frac{2\cdot 2\pi}{(2m+1)^2} & \text{if } \omega'_{k_1} \neq 2, \end{cases} \tag{16}$$

$$\beta_{left,t+1} = \left(\beta_{left,t} - \frac{2\pi\omega_{k_1-1}}{2m+1}\right)(2m+1). \tag{17}$$

2. If $1 < k_1 < k_2$:

$$\beta_{current,t+1} = \frac{2\pi\omega_{k_1-1}}{2m+1}, \tag{18}$$

$$\beta_{right,t+1} = \frac{\beta_{right,t}}{2m+1} + \frac{2\pi\omega'_{k_1}}{2m+1}, \tag{19}$$

$$\beta_{left,t+1} = \left(\beta_{left,t} - \frac{2\pi\omega_{k_1-1}}{2m+1}\right)(2m+1). \tag{20}$$

With selection controllers, the system can change the rotational positions of $D_{current}$, D_{left}, and D_{right} from $\beta_{current,t}$, $\beta_{left,t}$, and $\beta_{right,t}$ to $\beta_{current,t+1}$, $\beta_{left,t+1}$, and $\beta_{right,t+1}$, respectively, according to the recursive equations described above.

3.4. Putting it together. With mechanical components that can simulate the transition function and the tape of a universal TM, the frictional mechanical system that simulates the universal TM is almost immediate. The movements of the objects in this system are controlled by the rotations of the sequencer. In each cycle of rotation of the sequencer, the frictional mechanical system will finish the simulation of one step of the universal TM.

At the beginning of each cycle, the symbol at the current head location is decided, depending on the rotational position of $D_{current}$. According to the current symbol as well as the current state recorded by D_{state}, the control component will make a sequence of moves and then decide (i) the next state, (ii) the symbol replacing the current symbol, and (iii) the next move of the tape head. The tape component will then make a sequence of moves to change the rotational positions of $D_{current}$, D_{left}, and D_{right} according to the symbol replacing the current symbol and the next move. This finishes the simulation of one step of the computation of the universal TM.

Initially, D_{state} is set at the orientation encoding the start state of the TM. The orientations of $D_{current}$, D_{left}, and D_{right} also correspond to the initial status of the read-write tape. The simulation for the computation of the universal TM terminates when the rotational position of D_{state} is found to be corresponding to the accepting or rejecting state of the universal TM (we call these rotational positions *terminating orientations*) at the beginning of a rotational cycle of the sequencer.[5]

Hence, we have proved the following theorem.

THEOREM 3.1. *For any universal TM M, a frictional mechanical system can be constructed which has the property that, for any input string ω of M, the objects to the system can be set in a corresponding initial configuration so that a specified final configuration can be reached if and only if M accepts ω.*

As M is a universal TM, our frictional mechanical system can be used for general-purpose computing. Thus it has the computational power of any conventional electronic computer.

4. A frictional mechanical system with error. As mentioned in the introduction of this paper, our mechanical frictional system can simulate a universal TM without any error only if the system can be constructed and work exactly as it is specified. In the presence of errors, it is not possible to record the current configuration of the universal TM by the rotational positions of D_{state}, $D_{current}$, D_{left}, and D_{right} exactly as it is. Therefore, the computational power of the frictional mechanical system is restricted, as the errors in the rotational positions of these discs may be accumulated significant enough to induce an incorrect result in the simulation of the universal TM.

[5]Observe that D_{state} might be at a terminating orientation momentarily as it rotates around. However, if at the beginning of a rotational cycle D_{state} stays at a terminating orientation, this means that the universal TM has reached an accepting or rejecting state.

We use ϵ to denote the upper bound for the error in an angle which occurs in a single mechanical operation. ϵ is determined by the degree of accuracy in constructing mechanical devices as well as measuring rotational (and transitional) displacements.

4.1. A constant size frictional mechanical system in ϵ-error model. We first discuss a frictional mechanical system in this ϵ-error model; this system has exactly the same structure as the system in the exact model, and therefore it has only a constant number of parts.

Let $\beta'_{state,t}$ denote the correct angle of D_{state} at the beginning of the tth rotational cycle if the frictional mechanical system has no error. Further, let $\Delta_{state,t} =\mid \beta'_{state,t} - \beta_{state,t} \mid$; i.e., $\Delta_{state,t}$ is the error in $\beta_{state,t}$. $\beta'_{current,t}$, $\Delta_{current,t}$, $\beta'_{left,t}$, $\Delta_{left,t}$, $\beta'_{right,t}$, and $\Delta_{right,t}$ are defined accordingly.

In each rotational cycle, it takes a sequence of angle operations to generate each of $\beta_{state,t+1}$, $\beta_{current,t+1}$, $\beta_{left,t+1}$, and $\beta_{right,t+1}$ in the frictional mechanical system. Each operation involves adding an angle to (or deducting an angle from) the rotational position of a disc, multiplying an angle by a constant factor, or a nonlinear mapping. It is easy to see that there exist two constants c_1 and c_2 such that if the error in a rotational position is Δ, after any single operation, the error is bounded by $c_1\Delta + c_2\epsilon$. Further, as in each cycle the number of operations performed to get $\beta_{current,t+1}$, $\beta_{right,t+1}$, $\beta_{left,t+1}$, and $\beta_{state,t+1}$ is bounded by a constant, we can assume that there are two constants C_1 and C_2 such that $\Delta_{state,t+1} < C_1\Delta_{state,t} + C_2\epsilon$, $\Delta_{current,t+1} < C_1\Delta_{current,t} + C_2\epsilon$, $\Delta_{right,t+1} < C_1\Delta_{right,t} + C_2\epsilon$, and $\Delta_{left,t+1} < C_1\Delta_{left,t} + C_2\epsilon$.

For simplicity, we assume $C_1, C_2 > 2$.

There are only m valid values for $\beta'_{current,t+1}$ (i.e., $0, 2\pi/(2m+1), \ldots, 2(m-1)\pi/(2m+1)$), and the difference between any two valid values is at least $2\pi/(2m+1)$. Therefore, as long as $\Delta_{current,t+1} < \pi/(2(2m+1))$, a selection controller can be used to determine $\beta'_{current,t+1}$ from $\beta_{current,t+1}$. This means that $\beta_{current,t+1}$ can be corrected at the beginning of each cycle, given that $\epsilon < d_1/m$ for some constant $d_1 > 0$.

The same analysis applies to $\beta_{state,t+1}$: if $\epsilon < d_2/\sigma$ for some constant $d_2 > 0$, $\beta_{state,t+1}$ can be corrected at the beginning of each cycle. Therefore, only the errors in $\beta_{left,t}$ and $\beta_{right,t}$ may be accumulated to the next cycle. As at each step $\beta_{left,t}$ and $\beta_{right,t}$ are used only to distinguish ω_{k_1-1} and ω_{k_1+1}, errors in $\beta_{left,t}$ and $\beta_{right,t}$ will not cause any incorrectness if they are less than $\pi/(2(2m+1))$.

Observe that if the read-write head reaches the left end-mark, the error in $\beta_{left,t}$ will be discarded. This is because when the head moves rightward from the end-mark, none of $\beta_{current,t+1}$, $\beta_{right,t+1}$, and $\beta_{left,t+1}$ will depend on $\beta_{left,t}$, as shown in (9), (10), and (11). Similarly, if the read-write head reaches the right end-mark, the error in $\beta_{right,t}$ will be discarded too. Therefore, if the head of the universal TM visits the left and right end-marks periodically during the computation, the errors in β_{left} and β_{right} will be corrected periodically.

We define another TM M'. M' will simulate the computation of M. The difference between M' and M is that each time after M' finishes simulating M by a constant number K of steps of computation (this constant will be specified later), the head of M' will make a sweep of the tape. By "sweep" we mean that it will first move leftward until it reaches the left end-mark; then it will move rightward until it reaches the right end-mark; afterwards, it will move leftward again and return to the location where it was before the sweep. After it finishes the sweep, M' will start simulating the next K steps of the computation of M. It is easy to see that M' has the property that, at any time, if the current working space is s, the head of M' will reach the left

and right end-marks within at most $2S + 2K$ steps.

Clearly, M' is equivalent to M in the sense that it decides exactly the same language as M does. Therefore, to simulate M, it suffices to construct a frictional mechanical system that can simulate M'. For any input string ω of M', if M' decides ω in space S, the errors in β_{left} and β_{right} will be accumulated through at most $2S + 2K$ steps before they are discarded.

Thus, if the errors in β_{left} and β_{right} accumulated in $2S + 2K$ steps can be bounded by $\pi/(2(2m + 1))$ (otherwise incorrect values of ω_{k_1-1} and ω_{k_1+1} might be used), it will not induce any incorrect result in simulating the universal TM M' (and thus in simulating M) on input string ω.

Suppose at the tth step the read-write head moves rightward from the left end-mark. As the accumulated error of β_{left} is discarded at this moment, $\Delta_{left,t}$ is bounded by a constant, say, $C_3\epsilon$. As $\Delta_{left,t+1} < C_1\Delta_{left,t} + C_2\epsilon$, we have

$$\begin{aligned}\Delta_{left,t+2S+2K} &< C_1^{2S+2K}\Delta left, t + C_2\frac{C_1^{2S+2K} - 1}{C_1 - 1}\epsilon \\ &< C_1^{2S+2K}C_3\epsilon + C_2C_1^{2S+2K}\epsilon.\end{aligned}$$

To bound Δ_{left} by $\pi/(2(2m + 1))$, it suffices to let

$$\epsilon < \frac{1}{2(2m + 1)(C_3C_1^{2S+2K} + C_2C_1^{2S+2K})} < d \cdot 2^{-c(S+K)}$$

for some constants c and d.

If we let K be 1, M' will sweep the tape after each step of simulating M. Therefore, for any input string ω, it will take at most $(2S + 1)T$ steps for M' to decide ω if M decides ω in time T and space S. Since $S \leq T$, the total time used by M' is bounded by $(2T + 1)T$. Hence, we have the following theorem.

THEOREM 4.1. *For any universal TM M, a frictional mechanical system with error can be constructed to simulate M. It has the property that, for any space bound S, if the single-operation error of the system, ϵ, is bounded by* $\min\{d \cdot 2^{-cS}, d_1/m, d_2/\sigma\}$ *for some constants c, d, d_1, and d_2, then, given any input string ω that M decides in space bound S, the frictional mechanical system will reach a distinguished final configuration from an initial configuration encoding ω if and only if M accepts ω. Further, the frictional mechanical system will take at most $(2T + 1)T$ cycles to finish the computation if M decides ω in T steps.*

If we let $K = S$, M' will simulate M by S steps between two consecutive sweeps. Thus, it will take at most three times as much time as M takes to finish the computation. The disadvantage is that now M' (and hence our frictional mechanical system) depends on S. For different values of S, a different frictional mechanical system needs to be constructed to simulate M.

With some additional reasonable assumption, we can show that a periodic sweeping is not necessary. Given that $\epsilon = O(2^{-cS})$, if M can decide an input string ω in space S, our frictional mechanical system will finish simulating M with the correct result. Due to the length of the proof, we will include it in the appendix.

4.2. A frictional mechanical system in ϵ-error model with $\epsilon = \Omega(1)$. The analysis of the above system implies that $1/\epsilon$ will have to increase exponentially as S increases to maintain the correctness of the frictional mechanical system. If in a frictional mechanical system $O(S)$ discs instead of three discs are used to encode the contents of the tape, each representing one cell, this frictional mechanical system in

ϵ-error model can simulate the universal TM M with space bound S, where $\epsilon = \Omega(1)$. This is because now each disc has only a constant number of valid rotational positions, and selection controllers can be used to distinguish the correct value represented by the disc. This is better than the previous result in the sense that $1/\epsilon$ does not increase as S increases. However, the frictional mechanical system is now dependent on S, as it needs S discs to encode the tape. In other words, this model of frictional mechanical systems is a digital computer, just like the analytical engine. If we want to increase the computational power of the frictional mechanical system by using larger space bound S, we will have to add more discs used to represent the tape.

More specifically, for the kth cell of the tape, four discs are used to represent the current state of the cell: $D_{k,symbol}$ represents the symbol in the current cell; $D_{k,head}$ indicates whether the read-write head is located at this cell; $D'_{k,head}$ indicates whether the read-write head was located at this cell at the end of last step; and $D_{k,next}$ indicates the next move of the head. Also, for each cell, there is a set of nonlinear mapping devices which implement the transition function. We call all the parts used to represent the kth cell the kth cell component and denote it by C_k. In addition, there is a single disc D_{state} which records the current state of the universal TM.

At the beginning of each rotational cycle of the power disc, both $D_{k,head}$ and $D'_{k,head}$ are set at $\pi/4$ if the head is located at the kth cell or 0 if not. $D_{k,symbol}$ is set at $2i\pi/(2m+1)$ if the symbol in the kth cell is i. And D_{state} is set at $j\pi/(2\sigma)$, indicating that the current state of the TM is q_j. Then, if the rotational position of $D'_{k,head}$ is $\pi/4$, the rotational position of $D_{k,symbol}$ will be changed to $2i'\pi/(2m+1)$, where $i' = \delta_2(q_j, i)$. If the rotational position of $D'_{k,head}$ is 0, $D_{k,symbol}$ will be clamped so that its rotational position will not be changed. Also, the rotational position of $D_{k,next}$ is set at 0 if the next move is to go leftward and $\pi/4$ if the next move is to go rightward.

The only thing remaining is to change each $D_{k,head}$ accordingly. More specifically, for each k, if the rotational position of $D_{k,head}$ is $\pi/4$, (i.e., the head is at the kth cell), $D_{k-1,head}$ (or $D_{k+1,head}$) will be changed to $\pi/4$ if $D_{k,next}$ is set at 0 (or $\pi/4$, respectively) while $D_{k,head}$ will be reset to 0; if the rotational position of $D_{k,head}$ is 0, $D_{k,head}$, $D_{k-1,head}$, and $D_{k+1,head}$ will be clamped.

The problem is that there will be some conflicts in the movements of different components. For example, suppose the current head is located at the k_1th cell. Then, the k_1th component will try to change $D_{k_1-1,head}$ in case the next move is to go leftward. However, the (k_1-1)th component itself will try to clamp $D_{k_1-1,head}$ as the rotational position of $D'_{k_1-1,head}$ is 0. To resolve this problem, we divide the S head components into three groups so that C_k belongs to the (k mod 3)th group. First, the components in group 0 will be activated to change all the $D_{k,head}$s according to the rules described above. Then the components in group 1 will be activated. And last are the components in group 2. By this, no two components that have conflict movement will be activated at the same time.

After all components have been activated, for each k, $D_{k,head}$ will represent whether the read-write head is located at the kth cell at the beginning of the next step. Then the rotational position of $D_{k,head}$ is copied to $D'_{k,head}$ so that each component is ready for the simulation of the next computation of the universal TM.

It is easy to see that for a frictional mechanical system constructed as above, ϵ can be set to a constant regardless of the space bound S, as each disc has only a constant number of valid orientations and operations; only constant number of operations are needed to finish one step of simulation.

5. Reduction from A_{TM} to the frictional mover's problem. We prove the undecidability of the frictional mover's problem by a reduction from the A_{TM} problem. It suffices to show that every input of the A_{TM} problem, which is the description of a TM M along with an input string ω of M, can be transformed to an input of the frictional mover's problem, which is a frictional mechanical system with initial and goal configurations such that (i) the transformation can be performed by a specified procedure that terminates in a finite period of time and (ii) M accepts ω if and only if the frictional mechanical system can reach the goal configuration from its initial configuration.

Therefore, it is necessary that all the objects in the resulting frictional mechanical system and their positions can be described by rational coefficients. For example, for each disc, its radius as well as the location of its center need to be specified by rational numbers. Similarly, the length, orientation, and location of a bar need to be specified by rational numbers.

Recall that a universal frictional mechanical system involves only operations that add to a value (represented by the rotational position of a certain disc) a rational number or multiply the value by a rational number. Therefore, we can easily specify all the discs (including partial discs), bars, and cylinders in the system by rational coefficients without changing the functionalities of the various devices constructed by these parts.

The only exception is the irregular discs used in the nonlinear mapping devices. Each irregular disc has to be specified by some irrational coefficients, as indicated by (1). Here we prove the following lemma.

LEMMA 5.1. *The possibly irrational coefficients specifying the irregular discs used in the nonlinear mapping devices can be replaced by rational numbers without inducing any error in simulating the universal TM.*

Proof. Let D_1 be an irregular disc. It can be specified by the following two sets of parameters: $\{\theta_1, \theta_2, \ldots, \theta_n\}$, $\{r_1(\theta_0), r_1(\theta_1), \ldots, r_1(\theta_n)\}$. (Recall that $\theta_0 = 0$.) The boundary of D_1 inside the interval of $[\theta_i, \theta_{i+1}]$ is described by a linear function $r_1(\theta) = ((\theta - \theta_i) \cdot r_1(\theta_{i+1}) + (\theta_{i+1} - \theta) \cdot r_1(\theta_i))/(\theta_{i+1} - \theta_i)$, whose coefficients are entirely determined by $\theta_i, \theta_{i+1}, r_1(\theta_i)$, and $r_1(\theta_{i+1})$. Recall that in a universal frictional mechanical system, nonlinear mapping devices are used to implement the transition function δ, which takes only rational numbers as input. Therefore, $\theta_1, \theta_2, \ldots, \theta_n$ are all rational numbers. However, the exact value of each $r_1(\theta_i)$, denoted by $r_{1,i}$, could be an irrational number, as the computation involves irrational number $\sqrt{2}$ as well as the function tan. We replace each $r_{1,i}$ by a rational number $r'_{1,i}$ such that $|r'_{1,i} - r_{1,i}| < \epsilon'$ for some small $\epsilon' > 0$.

The nonlinear mapping device constructed with these "rationalized" parameters may cause two types of errors: *mechanical errors* and *logical errors*. A mechanical error occurs when two mechanical parts lose contact with, or crush into, each other while they are supposed to maintain continuous contact. A logic error occurs when the error of the resulting angle α causes a false interpretation by the universal frictional mechanical system.

We first show that mechanical errors can be avoided. We let D_2 be the coupling irregular disc of D_1 in the same nonlinear mapping device. We need to show that D_2 can also be described by rational numbers while keeping $r_1(\theta) + r_2(\theta + \pi)$ a constant for all $\theta \in [0, 2\pi]$. For each $\theta_i, i = 0, i, 2, \ldots, n$, the radius of D_2 at angle $\theta_i + \pi$ is computed by $r_2(\theta_i + \pi) = |\overline{O_1O_2}| - d - r_1(\theta_i)$. Since now $r_1(\theta_i)$ is chosen to be a rational number $r'_{1,i}$, and $|\overline{O_1O_2}|$ and d are all rational numbers, $r_2(\theta_i + \pi)$ is a rational

number. Therefore, the boundary of D_2 can be decomposed into $n+1$ subboundaries, corresponding to angular intervals $[\pi, \pi+\theta_1]$, $[\pi+\theta_1, \pi+\theta_2]$, ..., $[\pi+\theta_{n-1}, \pi+\theta_n]$, $[\theta_n - \pi, \pi]$, respectively. In each interval, the boundary can be specified by a linear function $r_2(\theta)$ of θ, just like the case for D_1. Therefore, D_2 can be described by rational numbers. For any $\theta \in (\theta_i, \theta_{i+1})$,

$$\begin{aligned} r_1(\theta) + r_2(\theta+\pi) &= \tfrac{(\theta-\theta_i)\cdot r_1(\theta_{i+1})+(\theta_{i+1}-\theta)\cdot r_1(\theta_i)}{\theta_{i+1}-\theta_i} + \tfrac{(\theta-\theta_i)\cdot r_2(\theta_{i+1}+\pi)+(\theta_{i+1}-\theta)\cdot r_2(\theta_i+\pi)}{\theta_{i+1}-\theta_i} \\ &= \tfrac{(\theta-\theta_i)(r_1(\theta_{i+1})+r_2(\theta_{i+1}+\pi))+(\theta_{i+1}-\theta)(r_1(\theta_i)+r_2(\theta_i+\pi))}{\theta_{i+1}-\theta_i} \\ &= |\overline{O_1O_2}| - d. \end{aligned}$$

This implies that irregular disc D_2 as described above still completely "complements" D_1 as defined in subsection 2.6. Therefore, during each operation D_1 and D_2 maintain continuous contact with the horizontal bar but do not compress it, thus causing no mechanical error.

Next we show that any mechanical error can be corrected. With the "rationalized" nonlinear mapping device, when the input angle is θ_i, the output angle is not exactly α_i but a value α_i' very close to α_i. Since nonlinear mapping devices are used to implement the transition function, each α_i is a rational number, as it encodes either a symbol or a state. Further, there exists a constant ϵ'' such that $|\alpha_i - \alpha_j| > \epsilon''$ if $\alpha_i \neq \alpha_j$. Here $= c \cdot \min\{\frac{1}{\sigma}, \frac{1}{m}\}$, where $c = \Omega(1)$. Therefore, there exists another constant $c' = \Omega(1)$ such that $|\alpha_i - \alpha_i'| < \frac{\epsilon''}{3}$ for all i if $\epsilon' < c' \cdot \min\{\frac{1}{\sigma}, \frac{1}{m}\} \cdot R$. Hence, a selection controller is able to find out the correct value of α_i from α_i', as α_i' is closer to α_i than to any other valid value.

This finishes the proof. □

With this lemma, we have the following theorem.

THEOREM 5.2. *The frictional mover's problem is undecidable.*

6. Conclusion. In this paper we introduced frictional mechanical systems and proved that a universal frictional mechanical system can simulate the computation of a universal TM. We also gave some results for the case where there are limited errors for the mechanical parts in the system. Our work implies that the frictional mover's problem is undecidable. It is, however, unclear to us what the implication of our work is to building nanocomputers at the macromolecule level when the nanotechnology further matures.

Appendix A. Error bound for β_{right} and β_{left} under additional assumption.

We have proved that by forcing the read-write head to visit the left and right end-marks periodically, the errors in β_{left} and β_{right} can be discarded after a certain number of steps so that they will not affect the result of simulating the universal TM. Here, we will show that, with the following assumption, there is no need for periodic sweeps of the tape.

Assumption 1. For any operation that will multiply an angle by a constant factor C, if before the operation the error in the angle is Δ, after the operation the error will become $C\Delta + k\epsilon$ for some constant k.

Under this assumption, an error in an angle θ will be reduced if the operation is to reduce θ by a constant factor (i.e., $C < 1$).

Now we will examine how the error in β_{right} is accumulated in each step using this assumption. We decompose $\Delta_{right,t}$ into two components: $\Delta^1_{right,t+1}$ and $\Delta^2_{right,t+1}$. The first component is due to $\Delta_{right,t}$, and the other component is the

error which occurs during the angle operations in this cycle. As in one cycle the number of operations to generate β_{right} is bounded by a constant number, we can assume $\Delta^2_{right,t+1} \leq C'_1\epsilon$ for some constant C'_1. $\Delta^1_{right,t+1}$, however, is determined by $\Delta_{right,t}$ as well as the angle operation on $\beta_{right,t}$ to generate $\beta_{right,t+1}$.

If the next move is rightward, according to (10) and (13), $\beta_{right,t}$ is multiplied by a factor of $(2m+1)$ and then added to another angle to get $\beta_{right,t+1}$. Therefore, the error in $\beta_{right,t}$ is amplified by a factor of $(2m+1)$, i.e., $\Delta^1_{right,t+1} = \Delta_{right,t} \cdot (2m+1)$. Similarly, if the next move is leftward, we have $\Delta^1_{right,t+1} = \Delta_{right,t}/(2m+1)$, as in (16) and (19) $\beta_{right,t+1}$ has a component $\beta_{right,t}/(2m+1)$.

Therefore, in each cycle, the error in β_{right} which is accumulated in previous cycles either is amplified by a factor of $(2m+1)$ or reduced to $1/(2m+1)$, depending on the move of the head. Hence we have the following recursive function:

$$\Delta_{right,t+1} \leq f(t)\Delta_{right,t} + \Delta^2_{right,t+1}. \tag{21}$$

Here $f(t)$ is defined as follows:

$$f(t) = \begin{cases} 2m+1 & \text{if the move of the head in the } t\text{th step is rightward,} \\ \frac{1}{2m+1} & \text{if the move of the head in the } t\text{th step is leftward.} \end{cases}$$

$\Delta_{right,1}$ is the error in β_{right} at the beginning of the computation. This error is due to the inaccuracy in setting the starting orientation of D_{right}. We can assume that $\Delta_{right,1} \leq C'_2\epsilon$.

According to the recursive equation, we can compute $\Delta_{right,t'+1}$ for any t':

$$\begin{aligned} & \Delta_{right,t'} \\ \leq\ & f(t'-1)\Delta_{right,t'-1} + \Delta^2_{right,t'} \\ \leq\ & f(t'-1)(f(t'-2)\Delta_{right,t'-2} + \Delta^2_{right,t'-1}) + \Delta^2_{right,t'} \\ & \vdots \\ & \vdots \\ \leq\ & \Delta_{right,1}\left(\prod_{i=1}^{t'-1} f(i)\right) + \sum_{j=2}^{t'} \Delta^2_{right,j}\left(\prod_{i=j}^{t'-1} f(i)\right). \end{aligned}$$

For any input string ω of M, we let S denote the working space M needs to use to decide ω. It is easy to see that, for any t' and j, $\prod_{i=j}^{t'-1} f(i) \leq (2m+1)^S$ as the right moves could outnumber the left moves at most by S. Therefore, $\Delta_{rught,t'}$ is bounded by

$$\Delta_{right,1}(2m+1)^S + \sum_{j=2}^{t'} \Delta^2_{right,j}(2m+1)^S \leq C'_2\epsilon(2m+1)^S + 2^{cS}C'_1\epsilon(2m+1)^S,$$

as $t' < 2^{cS}$ for some constant c.

To bound Δ_{right} by $1/(2(2m+1))$ (so that at each cycle correct value of ω_{k_1+1} could be retrieved from β_{right}), we need only

$$\epsilon \leq \frac{1}{2(2m+1)} \cdot \frac{1}{(\epsilon(2m+1)^S)(C'_2 + 2^{cS}C'_1)} < d' \cdot 2^{-c''S}$$

for some constants c'' and d'.

We have thus proved the following theorem.

THEOREM A.1. *For any universal TM M, a frictional mechanical system with error can be constructed to simulate M. It has the property that, for any space bound S, if the single-operation error of the system, ϵ, is bounded by* $\min(d' \cdot 2^{-c''S}, d_1/m, d_2/\sigma)$ *for some constants c'', d', d_1, and d_2, then, given any input string ω that M decides in space bound S, the frictional mechanical system will reach a distinguished final configuration from an initial configuration encoding ω if and only if M accepts ω. Further, the frictional mechanical system will take T cycles to finish the computation if M decides ω in T steps.*

REFERENCES

[1] T. ASANO, D. KIRKPATRICK, AND C. K. YAP, *d_1-optimal motion for a rod*, in Proceedings of the 12th Annual ACM Symposium on Computational Geometry, Philadelphia, PA, 1996, pp. 252–263.

[2] J. CANNY, *Some algebraic and geometric computations in PSPACE*, in Proceedings of the 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, 1988, pp. 460–467.

[3] J. CANNY AND J. H. REIF, *New lower bound techniques for robot motion planning problems*, in Proceedings of the 28th Annual Symposium on Foundations of Computer Science, Los Angeles, CA, 1987, pp. 49–60.

[4] P. C. CHEN AND Y. K. HWANG, *Practical path planning among movable obstacles*, in Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, CA, 1991, pp. 444–449.

[5] B. DACRE-WRIGHT, J.-P. LAUMOND, AND R. ALAMI, *Motion planning for a robot and a movable object amidst polygonal obstacles*, in Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France, 1992, pp. 2474–2480.

[6] M. ERDMANN, *On a representation of friction in configuration space*, Internat. J. Robotics Res., 13 (1994).

[7] J. HOPCROFT, D. JOSEPH, AND S. WHITESIDES, *Movement problems for 2-dimensional linkages*, SIAM J. Comput., 13 (1984), pp. 610–629.

[8] J. E. HOPCROFT, J. T. SCHWARTZ, AND M. SHARIR, *On the complexity of motion planning for multiple independent objects: PSPACE-hardness of the "warehouseman's problem,"* Internat. J. Robotics Res., 3 (1984), pp. 76–88.

[9] J. H. REIF, *Complexity of the mover's problem and generalizations*, in Proceedings of the 20th IEEE Symposium on Foundations of Computer Science, San Juan, PR, 1979, pp. 421–427.

[10] J. H. REIF AND M. SHARIR, *Motion planning in the presence of moving obstacles*, in Proceedings of the 26th Annual Symposium on Foundations of Computer Science, Portland, OR, 1985, pp. 144–154.

[11] J. H. REIF AND Z. SUN, *Movement planning in the presence of flows*, in Proceedings of the 7th International Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 2125, Springer, Berlin, 2001, pp. 450–461.

[12] J. H. REIF AND H. WANG, *The complexity of the two dimensional curvature-constrained shortest-path problem*, in Proceedings of the 3rd Workshop on Algorithmic Foundations of Robotics, Houston, TX, 1998, pp. 49–57.

[13] J. SELLEN, *Lower bounds for geometrical and physical problems*, SIAM J. Comput., 25 (1996), pp. 1231–1253.

[14] P. SPIRAKIS AND C. K. YAP, *Strong NP-hardness of moving many discs*, Inform. Process. Lett., 19 (1984), pp. 55–59.

[15] G. WILFONG, *Motion planning in the presence of movable obstacles*, in Proceedings of the 4th Annual ACM Symposium on Computational Geometry, Urbana-Champaign, IL, 1988, pp. 279–288.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1475–1487

COMPUTING ELEMENTARY SYMMETRIC POLYNOMIALS WITH A SUBPOLYNOMIAL NUMBER OF MULTIPLICATIONS*

VINCE GROLMUSZ†

Abstract. Elementary symmetric polynomials S_n^k are the building blocks of symmetric polynomials. In this work we prove that for constant k's, S_n^k modulo composite numbers $m = p_1p_2$ can be computed using only $n^{o(1)}$ multiplications if the coefficients of monomials $x_{i_1}x_{i_2}\cdots x_{i_k}$ are allowed to be 1 either mod p_1 or mod p_2 but not necessarily both. To the best of our knowledge, no previous result yielded even a sublinear (i.e., n^ε, $0 < \varepsilon < 1$) number of multiplications for similar tasks. Moreover, our algorithm fits in the model of the most restrictive depth-3 arithmetic circuits (homogeneous, multilinear, or the graph model). In contrast, by a lower bound of Nisan and Wigderson [*Comput. Complexity*, 6 (1997), pp. 217–234], any homogeneous depth-3 circuit needs size $\Omega((n/2k)^{k/2})$ for computing S_n^k modulo primes. Moreover, the number of multiplications in our algorithm remains sublinear while $k = O(\log\log n)$. Our results generalize for other nonprime-power composite moduli as well. The proof uses perfect hashing functions and the famous BBR polynomial of Barrington, Beigel, and Rudich.

Key words. symmetric polynomials, arithmetic circuits, algebraic algorithms, composite modulus

AMS subject classifications. 13P05, 94C05, 94C10

DOI. 10.1137/S009753970342465X

1. Introduction.

Surprising ideas sometimes lead to considerable improvements in algorithms even for the simplest computational tasks. Let us mention here the integer-multiplication algorithm of Karatsuba and Ofman [19] and the matrix-multiplication algorithm of Strassen [27].

A new field with surprising algorithms is quantum computing. The most famous and celebrated results are Shor's algorithm for integer factorization [25] and Grover's database-search algorithm [17].

Since realizable quantum computers can handle only very few bits today, there are no practical applications of these fascinating quantum algorithms.

Computations involving composite, nonprime-power moduli (say, 6), on the other hand, can actually be performed on any desktop PC; unfortunately, we have little evidence of the power or applicability of computations modulo composite numbers (see, e.g., the circuit given by Kahn and Meshulam [18], or the low-degree polynomial of Barrington, Beigel, and Rudich [3]).

One of the problems here is the interpretation of the output of the computation. Several functions are known to be hard if computed modulo a prime. If we compute the same function f with 0-1 values modulo 6, then it will also be computed modulo—say, 3—since $f(x) \equiv 1 \pmod 6 \implies f(x) \equiv 1 \pmod 3$ and $f(x) \equiv 0 \pmod 6 \implies f(x) \equiv 0 \pmod 3$; consequently, computing f this way cannot be easier mod 6 than mod 3. This difficulty is circumvented in a certain sense by the definition of the weak representation of Boolean functions by mod 6 polynomials, defined in [28] and [3].

*Received by the editors March 24, 2003; accepted for publication (in revised form) June 12, 2003; published electronically September 17, 2003. The author acknowledges the partial support of the István Széchenyi Fellowship, research grants EU FP5 IST FET IST-2001-32012, and an ETIK grant.
http://www.siam.org/journals/sicomp/32-6/42465.html

†Department of Computer Science, Eötvös University, Budapest, Pázmány P. stny. 1/C, H-1117 Budapest, Hungary (grolmusz@cs.elte.hu).

We will consider here another interpretation of the output, called a-strong representation (Definition 2). This definition will be more suitable for computations in which the output is a polynomial and not just a number.

Our goal is to compute this representation of the elementary symmetric polynomials

$$S_n^k = \sum_{\substack{I\subset\{1,2,\ldots,n\}\\ |I|=k}} \prod_{i\in I} x_i \tag{0}$$

modulo nonprime-power composite numbers with a much smaller number of multiplications than is possible over rationals or prime moduli.

Informally, the a-strong representation modulo 6 of the polynomial S_n^k is a polynomial of the form

$$\sum_{\substack{I\subset\{1,2,\ldots,n\}\\ |I|=k}} a_I \prod_{i\in I} x_i, \tag{1}$$

where either $a_I \equiv 1 \pmod 3$ or $a_I \equiv 1 \pmod 2$, but not necessarily both.

1.1. Our main result. Our main contribution here is an algorithm, using only $n^{o(1)}$ multiplications, computing—for constant k's—an a-strong representation of S_n^k of the form (1) modulo nonprime-power, composite integers (Theorems 4 and 7 and Corollaries 5 and 8).

1.1.1. Why do we count only the multiplications? In algebraic algorithms it is quite usual to count only the multiplications in a computation (for example, in the algorithm of Karatsuba and Ofman [19], or in the matrix-multiplication algorithms of Strassen [27] or Coppersmith and Winograd [5]). The reason for this is that the multiplication is *considered* to be a harder operation than the addition in most practical applications, and moreover, the multiplication is *proven* to be harder in most theoretical models of computation.

For example, computing the PARITY is reduced to computing the multiplication of two n-bit sequences, and, consequently, two n-bit sequences cannot be multiplied on a polynomial-size, constant-depth Boolean circuit [7], while it is well known that two n-bit sequences can be added in such a circuit.

1.2. Previous lower bounds. Most existing lower bounds were proven in the arithmetic circuit model of depth 3; circuits in this model are often called $\Sigma\Pi\Sigma$ circuits [22], [26].

$\Sigma\Pi\Sigma$ circuits perform computations of the following form:

$$\sum_{i=1}^{r} \prod_{j=1}^{s_i} (a_{ij1}x_1 + a_{ij2}x_2 + \cdots + a_{ijn}x_n + b_{ij}).$$

If all the $b_{ij} = 0$ and all the s_i's are the same number, then the circuit is called a homogeneous circuit; otherwise it is inhomogeneous. The size of the circuit is the number of gates in it: $1 + r + \sum_{i=1}^{r} s_i$.

A special class of homogeneous $\Sigma\Pi\Sigma$ circuits is called in [22] *the graph model*: here all $s_i = 2$ and all $a_{ij\ell}$ coefficients are equal to 1, and, moreover, the clauses of a product cannot contain the same variable twice. Consequently, such a product corresponds to a complete bipartite graph on the variables as vertices.

Note. Our algorithms fit into this very restrictive graph model of depth-3 circuits.

Graham and Pollak [8] asked how many edge-disjoint bipartite graphs can cover the edges of an n-vertex complete graph. They proved that $n-1$ bipartite graphs are sufficient and necessary. Later, Tverberg gave a very nice proof for this statement [30]. Having relaxed the disjointness property, Babai and Frankl [2] asked what is the minimum number of bipartite graphs, which covers every edge of an n-vertex complete graph by an odd multiplicity. Babai and Frankl proved that $(n-1)/2$ bipartite graphs are necessary. The optimum upper bound for the odd-cover was proved by Radhakrishnan, Sen, and Vishwanathan [22]. Radhakrishnan, Sen, and Vishwanathan also gave matching upper bounds for covers, when the off-diagonal elements of matrix M are covered by multiplicity 1 modulo a prime.

Nisan and Wigderson [21] showed that any homogeneous $\Sigma\Pi\Sigma$ circuit needs size $\Omega((n/2k)^{k/2})$ for computing S_n^k. This result shows that the homogeneous circuits are much weaker in computing elementary symmetric polynomials than the inhomogeneous ones. Nisan and Wigderson also examined bilinear and multilinear circuits in [21]. Note that the circuits in our constructions for $S_n^2(x,y)$ and for $S_n^k(x^1,x^2,\ldots,x^k)$ are also multilinear circuits.

We should note that exponential lower bounds were proved recently for simple functions for $\Sigma\Pi\Sigma$ circuits by Grigoriev and Razborov [10] and by Grigoriev and Karpinski [9].

Most recently, Raz and Shpilka got nice lower bound results for arithmetic circuits [24], and Raz [23] proved a $\Omega(n^2\log n)$ lower bound for matrix-multiplication in the model where the constants in the arithmetic circuits are bounded, solving a long-standing open problem.

1.3. Previous upper bounds. By a result of Ben-Or [26], every elementary symmetric polynomial S_n^k can be computed over fields by size-$O(n^2)$ inhomogeneous $\Sigma\Pi\Sigma$ circuits, using one-variable polynomial interpolation. Note that our construction with homogeneous circuits modulo nonprime-power composites beats Ben-Or's bound for k's less than $c\log\log n$ (for some positive c's).

In a somewhat related model, Muller and Preparata [20] showed that n-variable symmetric Boolean functions can be computed by depth-$O(\log n)$, size-$O(n)$ Boolean circuit.

To the best of the author's knowledge, there are no previous results concerning the computation of some form of the elementary symmetric polynomials even with a sublinear (i.e., n^{ε}, $0<\varepsilon<1$) number of multiplications, in any depth. Our results give a sub*polynomial* (that is, $n^{o(1)}$) number of multiplications even in the most restrictive depth-3 circuit model.

2. Alternative strong representation of polynomials. S_n^k can be naturally computed by $\binom{n}{k}$ product-gates by a homogeneous $\Sigma\Pi\Sigma$ circuit over any ring by the circuit of (0). One can save a little bit from the cost of this obvious construction (e.g., for $k=2n-1$ multiplications instead of $\binom{n}{2}$ is enough), but, as we already mentioned, by the result of Nisan and Wigderson [21], size $\Omega((n/2k)^{k/2})$ is needed to compute S_n^k on homogeneous $\Sigma\Pi\Sigma$ circuits.

It is quite plausible to think that if we change the nonzero coefficients of the monomials of S_n^k to some other nonzero coefficients, then the computational complexity of this modified polynomial will not be changed much: simply because even in the modified polynomial we would still need to generate the monomials with the nonzero coefficients somehow.

This intuition is verified by the next lemma (proven in the last section) in the case of finite fields.

LEMMA 1. *Suppose that a homogeneous* $\Sigma\Pi\Sigma$ *circuit computes polynomial*

$$g(x) = \sum_{\substack{I\subset\{1,2,\ldots,n\}\\ |I|=k}} a_I \prod_{i\in I} x_i$$

over the q *element field* F_q *with* u *gates, where* $a_I \neq 0$ *in* F_q. *Then* S_n^k *can be computed by a homogeneous* $\Sigma\Pi\Sigma$ *circuit of size* $O(u^{q-1})$.

From this lemma and from the $\Omega((n/2k)^{k/2})$-lower bound of Nisan and Wigderson [21] it is obvious that computing g over finite fields needs

$$\Omega((n/2k)^{\frac{k}{2(q-1)}})$$

multiplication gates.

Consequently, we cannot save much by computing g instead of S_n^k: if computing S_n^k needs polynomially many gates in n, then computing g still needs polynomially many gates in n (for any constant k).

Our main result is, however, that we *can save* much by computing certain strong representations of the elementary symmetric polynomials, say, over the modulo 15 integers, Z_{15}. More exactly, such representations can be computed by $\Sigma\Pi\Sigma$ circuits containing subpolynomially many multiplication gates. (We call a function $h(n)$ subpolynomial if for all $\varepsilon > 0$, $h(n) = O(n^{\varepsilon})$.)

Several authors (e.g., [28], [3]) defined the weak and strong representations of Boolean functions for integer moduli. Here we need the definition of a sort of strong representation of polynomials modulo composite numbers. We call this representation *alternative-strong* representation, abbreviated *a-strong representation*.

DEFINITION 2. *Let* m *be a composite number* $m = p_1^{e_1}p_2^{e_2}\cdots p_\ell^{e_\ell}$. *Let* Z_m *denote the ring of modulo* m *integers. Let* f *be a polynomial of* n *variables over* Z_m:

$$f(x_1, x_2, \ldots, x_n) = \sum_{I\in\{0,1,2,\ldots,d\}^n} a_I x_I,$$

where $a_I \in Z_m$, $x_I = \prod_{i=1}^n x_i^{\nu_i}$, *where* $I = \{\nu_1, \nu_2, \ldots, \nu_n\} \in \{0, 1, 2, \ldots, d\}^n$. *Then we say that*

$$g(x_1, x_2, \ldots, x_n) = \sum_{I\in\{0,1,2,\ldots,d\}^n} b_I x_I$$

is an a-strong representation *of* f *modulo* m *if*

$$\forall I \in \{0, 1, 2, \ldots, d\}^n \quad \exists j \in \{1, 2, \ldots, \ell\}, \quad a_I \equiv b_I \pmod{p_j^{e_j}},$$

and if for some i, $a_I \not\equiv b_I \pmod{p_i^{e_i}}$, *then* $b_I \equiv 0 \pmod{p_i^{e_i}}$.

Example. Let $m = 6$, and let $f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_1x_3$; then $g(x_1, x_2, x_3) = 3x_1x_2 + 4x_2x_3 + x_1x_3$ is an a-strong representation of f modulo 6.

Remarks on the definition of the a-strong representations.

- The requirements of Definition 2 for the coefficients are more restrictive than the requirements for the coefficients of f in Lemma 1, since the latter requires that the coefficients should not be zero, while the former prescribes that they should satisfy some congruencies.

- Definition 2 is a natural generalization of reducing a polynomial modulo an integer m. Usually, it can be done by reducing the coefficients modulo m, and when we compute the value of the polynomial for a given substitution, then the additions and multiplications should be done modulo m. If m has only one prime-divisor, then our definition and the usual one coincide. On the other hand, for example, $m = p_1^{\alpha_1} p_2^{\alpha_2}$ (p_1, p_2 are primes), and if we consider a polynomial P where all the coefficients are one (e.g., one of the elementary symmetric polynomials), then any computation which outputs P modulo m can also be viewed modulo p_1 or p_2, and also outputs P modulo p_1 or p_2. Consequently, by the usual definition of modulo m polynomials, one cannot compute any such P faster modulo a composite m than modulo a prime.
- The earlier (strong-, weak-)representations of functions [28], [3] contained constraints for the *value* of certain polynomials. Now we are requiring that the *form* of the representation satisfy modular constraints. The requirement on the form of the representation, in general, will not imply that the value corresponds in some transparent way to the represented polynomial. Let us consider the elementary symmetric polynomial $S_n^k(x)$, and suppose that polynomial $f(x)$ a-strongly represents $S_n^k(x)$ modulo 15. Then, for any $x' \in \{0,1\}^n$ which contains exactly k 1's, $f(x')$ will be 1 either mod 3 or mod 5; but, in general, $f(x)$ will not necessarily be even symmetric, and it can be 0 modulo 15 at points other than $S_n^k(x)$. On the other hand, every coefficient of the polynomial $S_n^k(x) - f(x)$ should be zero either mod 3 or mod 5.
- The a-strong representation is not unique; for example, polynomial $f(x)$ is always an a-strong representation of itself.
- In a sequel to the present work (available as a preprint [16]), we defined a complement of the a-strong representation of polynomials and showed how to compute such a representation of the dot-product of two length-n vectors with $n^{o(1)}$ multiplications, or the representation of the product of two $n \times n$ matrices with $n^{2+o(1)}$ multiplications.

Our goal in this work is to show that the elementary symmetric polynomials have a-strong representations modulo composites which can be computed by much smaller homogeneous $\Sigma\Pi\Sigma$ arithmetic circuits than the original polynomial.

Unfortunately, we cannot hope for such results for all multivariate polynomials, as is shown by the next theorem.

THEOREM 3. *Let*

$$f(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} x_i y_i$$

be the inner product function. Suppose that a $\Sigma\Pi\Sigma$ circuit computes an a-strong representation of f modulo 6. Then the circuit must have at least $\Omega(n)$ multiplication gates.

Proof. Let g be the a-strong representation of f. Then in g, at least half of the monomials $x_i y_i$ have coefficients equal to 1 modulo either 2 or 3. Without restricting the generality, let us assume that monomials $x_1 y_1, x_2 y_2, \ldots, x_{\lceil n/2 \rceil} y_{\lceil n/2 \rceil}$ have coefficients 1 modulo 3. When we compute g modulo 6 we will learn also the inner product of two vectors modulo 3, each consisting of the first $\lceil n/2 \rceil$ variables. It is well known that the communication complexity of computing the inner product mod 3 is $\Omega(n)$ (see, e.g., [11]).

Since arithmetic $\Sigma\Pi\Sigma$ circuits modulo 6 with u multiplication gates of in-degree 2

can be evaluated by a 2-party communication protocol using only $O(u)$ bits, we get $u = \Omega(n)$. □

3. Our constructions. First we construct a-strong representations with a small number of multiplications for the polynomial

$$S_n^2(x,y) = \sum_{\substack{i,j\in\{1,2,\ldots,n\}\\ i\neq j}} x_i y_j,$$

and for $x = y$ we will get that $2S_n^2(x) = S_n^2(x,x)$. This will imply our result for any composite, odd, nonprime-power moduli m.

THEOREM 4.

(i) *Let $m = p_1p_2$, where $p_1 \neq p_2$ are primes. Then there exists an a-strong representation of $S_n^2(x,y)$ modulo m,*

$$\sum_{\substack{i,j\in\{1,2,\ldots,n\}\\ i\neq j}} a_{ij}x_i y_j, \tag{2}$$

which can be computed on a homogeneous $\Sigma\Pi\Sigma$ circuit of size

$$\exp\left(O\left(\sqrt{\log n \log\log n}\right)\right).$$

Moreover, this representation satisfies that for all $i \neq j$, $a_{ij} = a_{ji}$.

(ii) *Let the prime decomposition of $m = p_1^{e_1}p_2^{e_2}\cdots p_r^{e_r}$. Then there exists an a-strong representation of $S_n^2(x,y)$ modulo m of the form (2) which can be computed on a homogeneous $\Sigma\Pi\Sigma$ circuit of size*

$$\exp\left(O\left(\sqrt[r]{\log n(\log\log n)^{r-1}}\right)\right).$$

Moreover, this representation satisfies that for all $i \neq j$, $a_{ij} = a_{ji}$.

COROLLARY 5.

(i) *Let $m = p_1p_2$, where $p_1 \neq p_2$ are odd primes. Then there exists an a-strong representation of the second elementary symmetric polynomial $S_n^2(x)$ modulo m which can be computed on a homogeneous $\Sigma\Pi\Sigma$ circuit of size*

$$\exp\left(O\left(\sqrt{\log n \log\log n}\right)\right).$$

(ii) *Let the prime decomposition of the odd m be $m = p_1^{e_1}p_2^{e_2}\cdots p_r^{e_r}$. Then there exists an a-strong representation of the second elementary symmetric polynomial $S_n^2(x)$ modulo m which can be computed on a homogeneous $\Sigma\Pi\Sigma$ circuit of size*

$$\exp\left(O\left(\sqrt[r]{\log n(\log\log n)^{r-1}}\right)\right).$$

Since the $\Sigma\Pi\Sigma$ circuit in our construction correspond to the graph model [22], we have the following graph-theoretical corollary, showing a cover with much fewer bipartite graphs than in the linear lower bound of Graham and Pollak.

COROLLARY 6. *For any $m = p_1^{e_1}p_2^{e_2}\cdots p_r^{e_r}$, there exists an explicitly constructible bipartite cover of the edges of the complete n-vertex graph such that for all edges e*

there exists an i, $1 \le i \le r$, that the number of the bipartite graphs covering e is congruent to 1 modulo $p_i^{e_i}$. Moreover, the total number of the bipartite graphs in the cover is

$$\exp\left(O\left(\sqrt[r]{\log n(\log\log n)^{r-1}}\right)\right).$$

3.1. Our results for larger k's. The following theorem gives our result for general k. Our goal is to compute an a-strong representation of polynomials $S_n^k(x)$ for $n \ge k \ge 2$. Let us first define

$$S_n^k(x^{(1)}, x^{(2)}, \dots, x^{(k)}) = \sum_{i_1, i_2, \dots, i_k} x_{i_1}^{(1)} x_{i_2}^{(2)} \cdots x_{i_k}^{(k)},$$

where the summation is done for all $k!$ orders of all k-element subsets $I = \{i_1, i_2, \dots, i_k\}$ of $\{1, 2, \dots, n\}$, and $x^{(j)} = (x_1^{(j)}, x_2^{(j)}, \dots, x_n^{(j)})$ for $j = 1, 2, \dots, k$.

THEOREM 7. *Let $m = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$. Then there exists an a-strong representation of $S_n^k(x^{(1)}, x^{(2)}, \dots, x^{(k)})$ modulo m,*

$$\sum_{i_1, i_2, \dots, i_k} a_{i_1, i_2, \dots, i_k} x_{i_1}^{(1)} x_{i_2}^{(2)} \cdots x_{i_k}^{(k)},$$

which can be computed on a homogeneous multilinear $\Sigma\Pi\Sigma$ circuit of size

$$\exp\left(\exp(O(k))\sqrt[r]{\log n}\log\log n\right).$$

Moreover, coefficients $a_{i_1, i_2, \dots, i_k}$ depend only on set $a\{i_1, i_2, \dots, i_k\}$ and not on the particular order of indices $i_1, i_2, \dots, i_k$.

Note that this circuit-size is subpolynomial in n for any constant k and for large enough n. Moreover, the subpolynomiality holds while $k < c \log\log n$ for a small enough $c > 0$.

For moduli m, relative prime to $k!$, this implies the following.

COROLLARY 8. *If m is a relative prime to $k!$, then there exists an a-strong representation of $S_n^k(x)$ modulo m which can be computed on a homogeneous $\Sigma\Pi\Sigma$ circuit of size*

$$\exp\left(\exp(O(k))\sqrt[r]{\log n}\log\log n\right).$$

3.2. The construction for computing S_n^2.

Proof of Theorem 4. We prove the more general case (ii) of the theorem.

Note that $S_n^2(x, y)$ contains the sum of the monomials $x_i y_j$ for all $i \ne j$. Let us arrange these monomials as follows: Let the x_i's and y_j's be assigned to the rows and columns of an $n \times n$ matrix M, respectively, and let the position in row i and column j contain monomial $x_i y_j$:

$$M = \begin{array}{c} \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & \cdots & y_n \end{array} \\ \begin{pmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_ny_1 & x_ny_2 & \cdots & x_ny_n \end{pmatrix} \end{array}. \tag{3}$$

Then any product of the form

$$(x_{i_1} + x_{i_2} + \cdots + x_{i_v})(y_{j_1} + y_{j_2} + \cdots + y_{j_w}) \tag{4}$$

naturally corresponds to a $v \times w$ submatrix of matrix M. We call these submatrices rectangles. Clearly, any a-strong representation modulo m of polynomial $S_n^2(x, y)$ can be obtained from a cover of matrix M by rectangles of the form (4), satisfying the following properties.

Property (a). The number of rectangles covering any elements of the diagonal is a multiple of $m = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$.

Property (b). Any nondiagonal element $x_i y_j$ of M is covered by d_{ij} rectangles, where

- there exists a $k \in \{1, 2, \ldots, r\}$: $d_{ij} \equiv 1 \pmod{p_k}$ and it is either 0 or 1 mod p_2;
- for all $k \in \{1, 2, \ldots, r\}$, $d_{ij} \equiv 0 \pmod{p_k}$ or $d_{ij} \equiv 1 \pmod{p_k}$.

Clearly, a (bilinear) $\Sigma\Pi\Sigma$ circuit computes an a-strong representation of polynomial $S_n^2(x, y)$ if and only if the corresponding rectangle cover satisfies Properties (a) and (b). The construction of such a low-cardinality rectangle cover is implicit in papers [12] and [13]. We present here a short direct proof which is easily generalizable for proving the results in the next section for higher dimensional matrices.

Rectangles, covering M, will be denoted

$$R(I, J) = \left(\sum_{i \in I} x_i\right)\left(\sum_{j \in J} y_j\right).$$

We now define an initial cover of the nondiagonal elements of M by rectangles.

Let $N = \lceil \log n \rceil$, and for $1 \le i, j \le n$, let $i = (i_1, i_2, \ldots, i_g)$ and $j = (j_1, j_2, \ldots, j_g)$ denote their N-ary forms (i.e., $0 \le i_t, j_t \le N - 1$ for $t = 1, 2, \ldots, g$, where $g = \lceil \log_N(n + 1) \rceil$).

Then let us define, for $t = 1, 2, \ldots, g$ and $\ell = 0, 1, \ldots, N - 1$,

$$I_t^\ell = \{i : i_t = \ell\}, \quad J_t^\ell = \{j : j_t \neq \ell\}.$$

Now consider the cover given by the following rectangles:

$$R(I_t^\ell, J_t^\ell) : t = 1, 2, \ldots, g, \ \ell = 0, 1, \ldots, N - 1.$$

Now, in this cover, any element $x_i y_j$ of M will be covered $H_N(i, j)$ times, where $H_N(i, j)$ stands for the Hamming distance of the N-ary forms of i and j, that is, at most g times. Note that the diagonal elements are not covered at all, so Property (a) is satisfied, while Property (b) is typically not. Moreover, $x_i y_j$ is covered by the same number of rectangles as $x_j y_i$, that is, $H_N(i, j)$ times.

The total number of covering rectangles is $h = gN = O((N \log n)/\log N)$.

Now, our goal is to turn this cover into another one, which already satisfies not only Property (a) but also Property (b). For this transformation we need to apply a multivariate polynomial f to our rectangle cover in a way very similar to how we applied polynomials to set-systems in [14] and to codes in [15].

DEFINITION 9. *Let $R_1, R_2, \ldots, R_h$ be a rectangle cover of a matrix $M = \{x_i y_j\}$, and let f be an h-variable multilinear polynomial written in the form*

$$f(z_1, z_2, \ldots, z_h) = \sum_{K \subset \{1, 2, \ldots, h\}} a_K z_K,$$

where $0 \le a_K \le m-1$ are integers, and $z_K = \prod_{k\in K} z_k$. Then the f-transformation of the rectangle cover $R_1R_2,\dots,R_h$ contains $\sum_{K\subset\{1,2,\dots,h\}} a_K$ rectangles, each corresponding to a monomial of f. $z_K = \prod_{k\in K} z_k$ correspond to the (possibly empty) rectangle of $\bigcap_{k\in K} R_k$.

Note that another way of interpreting this definition is as follows: the variables z_k correspond to the rectangles of the cover, and if we imagine the rectangles filled with 1's, then the product of the variables, i.e., the monomials, corresponds to the Hadamard product (see, e.g., [14]) of the corresponding all-1 rectangles, resulting in an all-1 rectangle, which, in turn, is equal to their intersection.

Note also that polynomial f is, in fact, considered over the ring Z_m, along with a fixed (small) representation of its coefficients from the set of nonnegative integers.

LEMMA 10. *Let $u^{ij} \in \{0,1\}^h$ characterize the rectangle cover of the entry x_iy_j of matrix M as follows:*

$$R_s \text{ covers } x_iy_j \iff u^{ij}_s = 1.$$

Then entry x_iy_j is covered by exactly $f(u^{ij})$ rectangles from the f-transformation of the rectangle cover $R_1, R_2, \dots, R_h$.

Proof. In $f(z)$, exactly those monomials z_K contribute 1 to the value of $f(u^{ij})$ whose variables are all 1 in vector u^{ij}. This happens exactly when $u^{ij}_k = 1$ for all $k \in K$; that is, x_iy_j is covered by the intersection of rectangles $\bigcap_{k\in K} R_k$. □

The proof of the following lemma is obvious.

LEMMA 11. *The intersection of finitely many rectangles is a (possibly empty) rectangle. Any rectangle, covering a part of matrix M of* (3)*, corresponds to a single (bilinear) multiplication.*

It remains to prove that there exists an f, with a small number of monomials, and with properties which lead to a cover, satisfying Properties (a) and (b). We will use the famous BBR polynomial of Barrington, Beigel, and Rudich [3].

THEOREM 12 (Barrington, Beigel, and Rudich [3]). *Let $m = p_1^{e_1}p_2^{e_2}\cdots p_r^{e_r}$. For any integers d, ℓ, $1 \le d \le \ell$, there exists an $f_{d,\ell}$ explicitly constructible, symmetric, ℓ-variable, degree-$O(d^{1/r})$ multilinear polynomial with coefficients from Z_m, such that*

(i) *for any $z \in \{0,1\}^\ell$, which contains at most d 1's,*

$$f_{d,\ell}(z) \equiv 0 \pmod m \iff z = 0;$$

(ii) *if $f_{d,\ell}(z) \not\equiv 0 \pmod m$, then there exists $i \in \{1,2,\dots,r\}$: $f_{d,\ell}(z) \equiv 1 \pmod{p_i^{e_i}}$, and if $f_{d,\ell}(z) \not\equiv 1 \pmod{p_j^{e_j}}$, then $f_{d,\ell}(z) \equiv 0 \pmod{p_j^{e_j}}$.*

Proof. (i) The proof of part (i) is given in [3] (see also [13]).

(ii) We consider $m = p_1^{e_1}p_2^{e_2}\cdots p_r^{e_r}$ to be a constant. Let us define $q_i = m/p_i^{e_i}$, and let $q_i^{-1}q_i \equiv 1 \pmod{p_i^{e_i}}$ for $i = 1,2,\dots,r$.

Let w denote the (symmetric) polynomial satisfying the requirements of (i).

Suppose first that $e_k = 1$ for $k = 1,2,\dots,r$. Then

$$f_{d,\ell} = \sum_{i=1}^{r} q_i q_i^{-1} w^{p_i-1}$$

is also symmetric and clearly satisfies the requirements of (ii). Indeed, if $w(z) \not\equiv 0 \pmod{p_i}$, then $f_{d,\ell}(z) \equiv 1 \pmod{p_i}$, and if $w(z) \equiv 0 \pmod{p_i}$, then $f_{d,\ell}(z) \equiv 0 \pmod{p_i}$. Moreover, the degrees of $f_{d,\ell}$ and w differ only in a constant multiplier.

In general, let us first consider the polynomial w which satisfies (i) for modulus $m' = p_1 p_2 \cdots p_r$. From the results of Toda [29], Yao [31], and Beigel and Tarui [4], for every k there exist polynomials P_k of degree $O(k)$ satisfying

$$\begin{aligned} P_k(x) &\equiv 0 \pmod{x^k}, \\ P_k(x+1) &\equiv 1 \pmod{x^k}. \end{aligned}$$

Now, let us define

$$f_{d,\ell} = \sum_{i=1}^{r} q_i q_i^{-1} P_{e_i}(w^{p_i - 1}).$$

It is easy to verify that (ii) is satisfied for this polynomial, and the degree is still $O(d^{1/r})$. Moreover, $f_{d,\ell}$ is also symmetric. □

Now we can prove Theorem 4; let us consider the more general statement of (ii). Let $\ell = h = gN$, $d = g$. Then $f_{g,gN}$ has

$$\binom{h}{O(g^{1/r})} \tag{5}$$

monomials. Consequently, if we transform our cardinality-h rectangle cover by Definition 9 with polynomial $f_{g,gN}$, then the resulting cover satisfies Properties (a) and (b) and has cardinality (5). This implies an $\exp(O(\sqrt[r]{\log n (\log\log n)^{r-1}}))$ cover. By Lemma 11, a $\Sigma\Pi\Sigma$ circuit is immediate with $\exp(O(\sqrt[r]{\log n (\log\log n)^{r-1}}))$ multiplication gates. Since the original, cardinality-h rectangle cover covered $x_i y_j$ and $x_j y_i$ with the same number of rectangles, and since $f_{g,gN}$ is a symmetric polynomial, by Lemma 10 our transformed rectangle cover will also cover $x_i y_j$ and $x_j y_i$ with the same number of rectangles. □

4. The construction in general. In this section we prove Theorem 7.

We describe a construction similarly to that of the case $k = 2$.

Note that in this section, instead of the more correct notation for vectors x with upper index u: $x^{(u)}$, we will write simply x^u.

First, let $M' = \{m_{i_1,i_2,\ldots,i_k}\}$ be a k-dimensional analogue of M of (3), that is, an

$$\overbrace{n \times n \times n \times \cdots \times n}^{k}$$

matrix, where $m_{i_1,i_2,\ldots,i_k} = x^1_{i_1} x^2_{i_2} \cdots x^k_{i_k}$.

Now we should again construct a cover of M', this time with k-dimensional boxes, corresponding to k-linear products,

$$R(I_1, I_2, \ldots, I_k) = \prod_{i=1}^{k} \sum_{j \in I_i} x^i_j,$$

satisfying that only those entries will be covered which have no two equal (lower) indices, and the covering multiplicity of these entries should be nonzero modulo m. Additionally, we also require that the covering multiplicity of entry $m_{i_1,i_2,\ldots,i_k}$ depend only on the set $\{i_1, i_2, \ldots, i_k\}$ and not on the particular order of the indices $i_1, i_2, \ldots, i_k$.

First we need to define an initial box cover of those entries of the k-dimensional matrix M' which have no two identical indices.

For our proof it is very important that this initial cover has low multiplicity: every covered element of M' should be covered only by $O(\log n)$ k-dimensional boxes for constant k's. The construction of such initial cover in the $k = 2$ case was quite easy; now we must use a more intricate approach.

Let us consider a family of perfect hash functions (see, e.g., [6] or the work [1] for an explicit (i.e., derandomized) construction), and let us list their respective values in the column of a matrix. This way, for integers n, k, b, $2 \le k \le b = O(k)$, $k \le n$, we can obtain a matrix $H(n,k,b) = \{h_{ij}\}$ with $u = \exp(O(k)) \log n$ rows and n columns, with entries from the set $\{0, 1, \dots, b-1\}$, such that for any k-element subset J of the n columns, there exists a row i, $1 \le i \le u$:

$$h_{ij}, \quad j \in J$$

are pairwise different elements of the set $\{0, 1, \dots, b-1\}$.

This matrix $H(n,k,b)$ will be used for the definition of our initial cover as follows:

For any i, $1 \le i \le u$, and for any $\sigma : \{1, 2, \dots, k\} \to \{0, 1, \dots, b-1\}$ injective function we define the k-dimensional box

$$R(i,\sigma) = \{m_{j_1,j_2,\dots,j_k} : h_{ij_1} = \sigma(1), h_{ij_2} = \sigma(2), \dots, h_{ij_k} = \sigma(k)\}.$$

There are u possible i's and $k^{O(k)}$ possible σ's, so there are $k^{O(k)} \log n$ boxes in this cover. Box $R(i,\sigma)$ covers only $m_{j_1,j_2,\dots,j_k}$'s with pairwise different indices.

It is important to note that even for a fixed i, the covering multiplicities of the elements $m_{j_1,j_2,\dots,j_k}$ and $m_{\pi(j_1)\pi(j_2)\dots,\pi(j_k)}$ are the same for any permutations π of the numbers $\{j_1, j_2, \dots, j_k\}$.

Any $m_{j_1,j_2,\dots,j_k}$ with pairwise different indices is covered by exactly as many k-dimensional boxes from this cover as the number of rows with pairwise different elements of the submatrix, containing column j_1, column j_2, …, column j_k of matrix $H(n,k,b)$. This number is at least 1 (from the perfect-hashing property) and at most u (that is, the number of rows of $H(n,k,b)$).

Now, exactly as in the proof of the S_n^2 case, we would like to apply the polynomial $f_{d,\ell}$ of Theorem 12 with $d = u$, $\ell = k^{O(k)} \log n$ to this box cover.

However, first we need to give the higher-dimension analogues of Definition 9 and Lemma 10.

DEFINITION 13. *Let $R_1, R_2, \dots, R_h$ be a box cover of a matrix M', and let f be an h-variable multilinear polynomial written in the form*

$$f(z_1, z_2, \dots, z_h) = \sum_{K \subset \{1,2,\dots,h\}} a_K z_K,$$

where $0 \le a_K \le m-1$ are integers, and $z_K = \prod_{k \in K} z_k$. Then the f-transformation of the box cover $R_1 R_2, \dots, R_h$ contains $\sum_{K \subset \{1,2,\dots,h\}} a_K$ boxes, each corresponding to a monomial of f. $z_K = \prod_{k \in K} z_k$ corresponds to the (possibly empty) box of $\bigcap_{k \in K} R_k$.

LEMMA 14. *Let $u^{i_1,i_2,\dots,i_k} \in \{0,1\}^h$ characterize the box cover of the entry $m_{i_1,i_2,\dots,i_k}$ of matrix M' as follows:*

$$R_s \text{ covers } m_{i_1,i_2,\dots,i_k} \iff u_s^{i_1,i_2,\dots,i_k} = 1.$$

Then entry $m_{i_1,i_2,\dots,i_k} = x_{i_1}^1 x_{i_2}^2 \cdots x_{i_k}^k$ is covered by exactly $f(u^{i_1,i_2,\dots,i_k})$ boxes from the f-transformation of the box cover $R_1, R_2, \dots, R_h$.

Proof. In $f(z)$, exactly those monomials z_K contribute 1 to the value of $f(u^{i_1,i_2,\ldots,i_k})$ whose variables are all 1 in vector $u^{i_1,i_2,\ldots,i_k}$. This happens exactly when $u_s^{i_1,i_2,\ldots,i_k} = 1$ for all $s \in K$; that is, $m_{i_1,i_2,\ldots,i_k} = x_{i_1}^1 x_{i_2}^2 \cdots x_{i_k}^k$ is covered by the intersection of boxes $\bigcap_{k\in K} R_k$. □

Note that for any symmetric polynomial f and any box cover which has covering multiplicity on $m_{i_1,i_2,\ldots,i_k}$, depending only on set $\{i_1, i_2, \ldots, i_k\}$, the f-transformation of the cover will also have the same multiplicity on $m_{j_1,j_2,\ldots,j_k}$ and on $m_{\pi(j_1)\pi(j_2)\ldots,\pi(j_k)}$ for any permutations π of the numbers $\{j_1, j_2, \ldots, j_k\}$.

The proof of the following lemma is obvious.

LEMMA 15. *The intersection of finitely many boxes is a (possibly empty) box. Any box covering a part of matrix M' corresponds to a single (multilinear) product.*

The result of applying $f_{d,\ell}$ with $d = u$, $\ell = k^{O(k)} \log n$ to our initial box cover of cardinality ℓ is a box cover of cardinality

$$\exp(\exp(O(k))(\log n)^{1/r} \log\log n),$$

proving Theorem 7. □

4.1. Proof of Lemma 1. Let $R_1, R_2, \ldots, R_h$ be the covering boxes defined by the homogeneous ΣΠΣ circuit. Let us remark that every degree-k monomial $\prod_{i\in I} x_i$ with pairwise different indices is covered by $a_I \neq 0$ boxes in this cover. In F_q, for any nonzero element s, $s^{q-1} = 1$. Now, let us apply polynomial

$$f(z_1, z_2, \ldots, z_h) = (z_1 + z_2 + \cdots + z_h)^{q-1}$$

to the box cover $R_1, R_2, \ldots, R_h$ according to Definition 13. Then, by Lemma 14, the covering multiplicity of the degree-k monomials $\prod_{i\in I} x_i$ with pairwise different indices will be 1 in F_q, while all the others will remain 0. That is, the corresponding ΣΠΣ circuit computes S_n^k over F_q. □

Acknowledgment. The author is grateful to Gábor Tardos for discussions on the subject.

REFERENCES

[1] N. ALON AND M. NAOR, *Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions*, Algorithmica, 16 (1996), pp. 434–449.

[2] L. BABAI AND P. FRANKL, *Linear Algebra Methods in Combinatorics*, Department of Computer Science, University of Chicago, 1992, preliminary version.

[3] D. A. M. BARRINGTON, R. BEIGEL, AND S. RUDICH, *Representing Boolean functions as polynomials modulo composite numbers*, Comput. Complexity, 4 (1994), pp. 367–382.

[4] R. BEIGEL AND J. TARUI, *On ACC*, Comput. Complexity, 4 (1994), pp. 350–366.

[5] D. COPPERSMITH AND S. WINOGRAD, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[6] M. L. FREDMAN AND J. KOMLÓS, *On the size of separating systems and families of perfect hash functions*, SIAM J. Algebraic Discrete Methods, 5 (1984), pp. 61–68.

[7] M. L. FURST, J. B. SAXE, AND M. SIPSER, *Parity, circuits and the polynomial time hierarchy*, Math. Systems Theory, 17 (1984), pp. 13–27.

[8] R. GRAHAM AND H. POLLAK, *On embedding graphs in squashed cubes*, in Graph Theory and Applications, Lecture Notes in Math. 303, Springer-Verlag, Berlin, 1972, pp. 99–110.

[9] D. GRIGORIEV AND M. KARPINSKI, *An exponential lower bound for depth* 3 *arithmetic circuits*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1998, pp. 577–582.

[10] D. GRIGORIEV AND A. A. RAZBOROV, *Exponential lower bounds for depth* 3 *arithmetic circuits in algebras of functions over finite fields*, Appl. Algebra Engrg. Comm. Comput., 10 (2000), pp. 465–487.

[11] V. Grolmusz, *Separating the communication complexities of MOD m and MOD p circuits*, J. Comput. System Sci., 51 (1995), pp. 307–313.
[12] V. Grolmusz, *Low-rank co-diagonal matrices and Ramsey graphs*, Electron. J. Combin., 7 (2000), research paper 15; available online from http://www.combinatorics.org.
[13] V. Grolmusz, *Superpolynomial size set-systems with restricted intersections mod* 6 *and explicit Ramsey graphs*, Combinatorica, 20 (2000), pp. 73–88.
[14] V. Grolmusz, *Constructing set-systems with prescribed intersection sizes*, J. Algorithms, 44 (2002), pp. 321–337.
[15] V. Grolmusz, *Pairs of Codes with Prescribed Hamming Distances and Coincidences*, Tech. Report DIMACS TR 2002-9, Center for Discrete Mathematics and Theoretical Computer Science, Piscataway, NJ, 2002; available online from ftp://dimacs.rutgers.edu/pub/dimacs/TechnicalReports/TechReports/2002/2002-09.ps.gz.
[16] V. Grolmusz, *Near Quadratic Matrix Multiplication Modulo Composites*, Tech. Report TR03-001, Electronic Colloquium of Computational Complexity, 2003; available online from ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/2003/TR03-001/index.html.
[17] L. K. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1996, pp. 212–219.
[18] J. Kahn and R. Meshulam, *On mod p transversals*, Combinatorica, 10 (1991), pp. 17–22.
[19] A. Karatsuba and Y. Ofman, *Multiplication of multidigit numbers on automata*, Soviet Phys. Dokl., 7 (1963), pp. 595–596.
[20] D. E. Muller and F. P. Preparata, *Bounds to complexities of networks for sorting and for switching*, J. ACM, 22 (1975), pp. 195–201.
[21] N. Nisan and A. Wigderson, *Lower bounds on arithmetic circuits via partial derivatives*, Comput. Complexity, 6 (1997), pp. 217–234.
[22] J. Radhakrishnan, P. Sen, and S. Vishwanathan, *Depth-*3 *arithmetic circuits for* $s_n^2(x)$ *and extensions of the Graham-Pollack theorem*, in Proceedings of FSTTCS, New Delhi, India, Lecture Notes in Comput. Sci. 1974, Springer-Verlag, Berlin, 2000, pp. 176–187.
[23] R. Raz, *On the complexity of matrix product*, in Proceedings of the 34th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 2002, pp. 144–151.
[24] R. Raz and A. Shpilka, *Lower bounds for matrix product, in bounded depth circuits with arbitrary gates*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, ACM Press, New York, 2001, pp. 409–418.
[25] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.
[26] A. Shpilka, *Lower Bounds for Small Depth Arithmetic and Boolean Circuits*, Ph.D. thesis, Hebrew University, Jerusalem, Israel, 2001; available online from http://www.cs.huji.ac.il/~amirs/publications/my_main.ps.gz.
[27] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.
[28] G. Tardos and D. A. M. Barrington, *A lower bound on the MOD* 6 *degree of the OR function*, Comput. Complexity, 7 (1998), pp. 99–108.
[29] S. Toda, *PP is as hard as the polynomial-time hierarchy*, SIAM J. Comput., 20 (1991), pp. 865–877.
[30] H. Tverberg, *On the decomposition of* K_n *into complete bipartite graphs*, J. Graph Theory, 6 (1982), pp. 493–494.
[31] A. C. Yao, *On ACC and threshold circuits*, in Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 619–627.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1488–1508

OPTIMAL EXTERNAL MEMORY INTERVAL MANAGEMENT*

LARS ARGE† AND JEFFREY SCOTT VITTER‡

Abstract. In this paper we present the external interval tree, an optimal external memory data structure for answering stabbing queries on a set of dynamically maintained intervals. The external interval tree can be used in an optimal solution to the dynamic interval management problem, which is a central problem for object-oriented and temporal databases and for constraint logic programming. Part of the structure uses a weight-balancing technique for efficient worst-case manipulation of balanced trees, which is of independent interest. The external interval tree, as well as our new balancing technique, have recently been used to develop several efficient external data structures.

Key words. interval management, stabbing queries, I/O efficient, data structures

AMS subject classifications. 68P05, 68P10, 68P15

DOI. 10.1137/S009753970240481X

1. Introduction.

In recent years external memory data structures have been developed for a wide range of applications, including spatial, temporal, and object-oriented databases and geographic information systems. Often the amount of data manipulated in such applications is too large to fit in main memory, and the data must reside on disk. In such cases the input/output (I/O) communication between main memory and disk can become a bottleneck. In this paper we develop an I/O-optimal and space-optimal external interval tree data structure for answering stabbing queries among a changing set of intervals. The structure is the central part of an optimal solution to the dynamic interval management problem.

1.1. Memory model and previous results.

We will be working in the standard model for external memory with one (logical) disk [31, 4]. We assume that each external memory access (called an *I/O operation* or just *I/O*) transmits one page of B elements. We measure the efficiency of an algorithm in terms of the number of I/Os it performs and the number of disk blocks it uses.[1]

The *dynamic interval management problem* is the problem of maintaining a set of intervals such that, given a query interval I_q, all intervals intersecting I_q can be reported efficiently. As discussed in [29, 30, 38], the problem is crucial for indexing constraints in constraint databases and in temporal databases. The key component of dynamic interval management is the ability to answer *stabbing queries* [30]. Given a set of intervals, a stabbing query with a point q asks for all intervals containing q.

*Received by the editors April 1, 2002; accepted for publication (in revised form) July 14, 2003; published electronically September 17, 2003. An extended abstract version of this paper was presented at the 1996 IEEE Symposium on Foundations of Computer Science (FOCS'96) [11].

http://www.siam.org/journals/sicomp/32-6/40481.html

†Department of Computer Science, Duke University, Durham, NC 27708 (large@cs.duke.edu). This author was supported in part by the ESPRIT Long Term Research Programme of the EU under project 20244 (ALCOM–IT) and by the National Science Foundation through CAREER grant CCR–9984099. Part of this work was done while this author was with BRICS, Department of Computer Science, University of Aarhus, Denmark.

‡Department of Computer Sciences, Purdue University, West Lafayette, IN 47907 (jsv@purdue.edu). This author was supported in part by the National Science Foundation under grants CCR–9522047 and CCR–9877133 and by the U.S. Army Research Office under grant DAAH04–96–1–0013. This work was done while this author was at Duke University.

[1]Often it is assumed that the main memory is capable of holding $\Omega(B^2)$ elements, that is, $\Omega(B)$ blocks. The structures developed in this paper work without this assumption.

By representing an interval $[x, y]$ as the point (x, y) in the plane, a stabbing query reduces to the special case of 2-sided 2-dimensional range searching called a *diagonal corner query*. In the diagonal corner query problem a set of points in the plane above the diagonal line $x = y$ should be stored such that, given a query point (q, q), all points (x, y) with $x \leq q$ and $y \geq q$ can be reported efficiently. The problem of 2-dimensional range searching has been the subject of much research. While B-trees and their variants [12, 21] have been an unqualified success in supporting 1-dimensional external range searching, they are inefficient at handling higher-dimensional problems. In internal memory many worst-case efficient structures have been proposed for 2-dimensional and higher-dimensional range search; see [3] for a survey. Unfortunately, most of these structures are not efficient when mapped to external memory. The practical need for I/O support has also led to the development of a large number of external data structures that do not have good theoretical worst-case update and query I/O bounds but do have good average-case behavior for common problems; see [24, 35] for surveys. The worst-case performance of these data structures is much worse than the optimal bounds achievable for dynamic external 1-dimensional range search using a B-tree.

Prior to the development of the structure presented in this paper, a number of attempts had been made to solve the external stabbing query problem. Kanellakis et al. [30] developed the metablock tree for answering diagonal corner queries in optimal $O(\log_B N + T/B)$ I/Os using optimal $O(N/B)$ blocks of external memory. Here T denotes the number of points reported. The structure supports insertions only in $O(\log_B N + (\log_B^2 N)/B)$ I/Os amortized. A simpler static structure with the same bounds was described by Ramaswamy [37]. In internal memory, the priority search tree of McCreight [32] can be used to answer more general queries than diagonal corner queries, namely 3-sided range queries, and a number of attempts have been made at externalizing this structure [16, 28, 39]. The structure by Icking, Klein, and Ottoman [28] uses optimal space but answers queries in $O(\log_2 N + T/B)$ I/Os. The structure by Blankenagel and Güting [16] also uses optimal space but answers queries in $O(\log_B N+T)$ I/Os (see also [14]). In both papers a number of nonoptimal dynamic versions of the structures are also developed. Ramaswamy and Subramanian [39] developed a technique called *path caching* for transforming an efficient internal memory data structure into an I/O-efficient structure. Using this technique on the priority search tree results in a structure that can be used to answer 2-sided queries, which are more general than diagonal corner queries but less general than 3-sided queries. This structure answers queries in the optimal $O(\log_B N+T/B)$ I/Os and supports updates in amortized $O(\log_B N)$ I/Os but uses nonoptimal $O((N/B)\log_2\log_2 B)$ space. Various other external data structures for answering 3-sided queries are also developed in [30] and [39]. Subramanian and Ramaswamy also designed the p-range tree for answering 3-sided queries [40]. The structure uses linear space, answers queries in $O(\log_B N + T/B + IL^*(B))$ I/Os, and supports updates in $O(\log_B N + (\log_B^2 N)/B)$ I/Os amortized. ($IL^*(\cdot)$ denotes the iterated $\log^*$ function, that is, the number of times $\log^*$ must be applied to get below 2). Finally, following the publication of the extended abstract version of this paper (and based on the results in this paper), Arge, Samoladas, and Vitter [8] developed an optimal external priority search tree, immediately implying an optimal stabbing query structure. Several structures have also been developed for the general 2- and higher-dimensional range searching problem, as well as for several other related problems. See [5, 6, 41] for surveys.

TABLE 1.1
Comparison of our data structure for stabbing queries with other data structures.

	Space (blocks)	Query I/O bound	Update I/O bound
Pri. search tree [28]	$O(\frac{N}{B})$	$O(\log_2 N + T/B)$	
XP-tree [16]	$O(\frac{N}{B})$	$O(\log_B N + T)$	
[37]	$O(\frac{N}{B})$	$O(\log_B N + T/B)$	
Metablock tree [30]	$O(\frac{N}{B})$	$O(\log_B N + T/B)$	$O(\log_B N + (\log_B N)^2/B)$ amortized (inserts only)
P-range tree [40]	$O(\frac{N}{B})$	$O(\log_B N + T/B + IL^*(B))$	$O(\log_B N + (\log_B N)^2/B)$ amortized
Path caching [39]	$O(\frac{N}{B}\log_2\log_2 B)$	$O(\log_B N + T/B)$	$O(\log_B N)$ amortized
Our result [11] ([8])	$O(\frac{N}{B})$	$O(\log_B N + T/B)$	$O(\log_B N)$

1.2. Overview of our results. The main contribution of this paper is an optimal external memory data structure for the stabbing query problem. As mentioned, our data structure gives an optimal solution to the interval management problem, and thus it settles an open problem highlighted in [30, 39, 40]. The structure uses $O(N/B)$ disk blocks to maintain a set of N intervals such that insertions and deletions can be performed in $O(\log_B N)$ I/Os and such that stabbing queries can be answered in $O(\log_B N + T/B)$ I/Os. In Table 1.1 we compare our result with previous solutions. Unlike previous nonoptimal structures, the update I/O bounds for our data structure are worst-case, and our structure works without assuming that the internal memory is capable of holding $\Omega(B^2)$ elements. Our structure is significantly different from the recently developed external priority search tree [8] and is probably of greater practical interest since it uses relatively fewer random I/Os when answering a query. Most disk systems are optimized for sequential I/O, and, consequently, random I/Os often take a much longer time than sequential I/Os.

Our solution to the stabbing query problem is an external version of the interval tree [22, 23]. In section 2, we present the basic structure, where the endpoints of the intervals stored in the structure belong to a fixed set of N points. In section 3, we then remove this "fixed endpoint-set assumption." In internal memory, the assumption is normally removed using a BB[α]-tree [34] as the base search tree structure [33], and this leads to amortized update bounds. However, as BB[α]-trees are unsuitable for implementation in external memory, we develop a new *weight-balanced B-tree* for use in external memory. This structure resembles the *k-fold tree* of Willard [43]. Like in internal memory, the use of a weight-balanced B-tree as the base tree results in amortized update bounds. In section 4, we then show how to remove the amortization from the structure.

Our external interval tree has been used to develop I/O-efficient structures for dynamic point location [1, 9].[2] It has also been used in several visualization applications [18, 19, 20]. Our weight-balanced B-tree has also found several other applications. In internal memory it can, for example, be used to convert amortized bounds to worst-case bounds. (Fixing B to a constant in our result yields an internal-memory interval tree with worst-case update bounds.) It can also be used as a (simpler) alternative to the rather complicated structure developed in [42] in order to add range restriction capabilities to internal-memory dynamic data structures. (It seems possible to

[2] Even though the external priority search tree [8] solves a more general problem than the external interval tree, it cannot be used as an alternative to the external interval tree in the point location structures.

use the techniques in [42] to remove the amortization from the update bound of the internal interval tree, but our method is much simpler.) In external memory, it has been used in the recently developed optimal external priority search tree [8], as well as in numerous other structures (e.g., [25, 26, 13, 1, 9]).

Finally, in section 5, we discuss how to use the ideas utilized in our external interval tree to develop an external segment tree using $O((N/B)\log_B N)$ space. This improves upon previously known external segment tree structures, which use $O((N/B)\log_2 N)$ disk blocks [15, 39].

2. External memory interval tree with fixed endpoint set. In this section, we present our external interval tree structure, assuming that the endpoints of the intervals stored in the structure belong to a fixed set E of size N. We also assume that the internal memory is capable of holding $O(B)$ blocks. We remove these assumptions in sections 3 and 4.

2.1. Preliminaries. Our external interval tree makes extensive use of two kinds of auxiliary structures: the B-tree [12, 21] and the "corner structure" [30]. B-trees, or more generally (a, b)-trees [27], are search tree structures suitable for external memory.

LEMMA 2.1. *A set of N elements can be stored in a B-tree structure using $O(N/B)$ disk blocks such that updates and queries can be performed in $O(\log_B N)$ I/Os. The T smallest (largest) elements can be reported in $O(T/B+1)$ I/Os. Given N sorted elements a B-tree can be built in $O(N/B)$ I/Os.*

A "corner structure" [30] is a data structure that can be used to answer stabbing queries on $O(B^2)$ intervals.

LEMMA 2.2. *(Kanellakis et al. [30]) A set of $K \le B^2$ intervals can be stored in an external data structure using $O(K/B)$ disk blocks such that a stabbing query can be answered in $O(T/B+1)$ I/Os, where T is the number of reported intervals.*

As discussed in [30], the corner structure can easily be made dynamic: updates are inserted into an update block, and the structure is rebuilt using $O(B)$ I/Os once B updates have been performed. The rebuilding is performed simply by loading the structure into internal memory, rebuilding it, and writing it back to external memory.

LEMMA 2.3. *Assuming $M \ge B^2$, a set of $K \le B^2$ intervals can be stored in an external data structure using $O(K/B)$ disk blocks such that a stabbing query can be answered in $O(T/B+1)$ I/Os and such that an update can be performed in $O(1)$ I/Os amortized. The structure can be constructed in $O(K/B)$ I/Os.*

In section 4.2 (where it will become clearer why the structure is called a "corner structure"), we show how the update bound can be made worst-case. In the process we also remove the assumption on the size of the internal memory.

2.2. The structure. An internal memory interval tree consists of a binary base tree on the sorted set of endpoints E, with the intervals stored in secondary structures associated with internal nodes of the tree [22]. An interval X_v consisting of all endpoints below v is associated with each internal node v in a natural way. The interval X_r of the root r is thus divided in two by the intervals X_{v_l} and X_{v_r} associated with its two children, v_l and v_r, and an interval is stored in r if it contains the "boundary" between X_{v_l} and X_{v_r} (if it overlaps both X_{v_l} and X_{v_r}). Intervals on the left (right) side of the boundary are stored recursively in the subtree rooted in v_l (v_r). Intervals in r are stored in two structures: a search tree sorted according to left endpoints of the intervals and one sorted according to right endpoints. A stabbing query with q is answered by reporting the intervals in r containing q and recursively reporting the relevant intervals in the subtree containing q. If q is contained in X_{v_l},

the intervals in r containing q are found by traversing the intervals in r sorted according to left endpoints, from the intervals with smallest left endpoints toward the ones with largest left endpoints, until an interval not containing q is encountered. None of the intervals in the sorted order after this interval can contain q. Since $O(T_r)$ time is used to report T_r intervals in r, a query is answered in $O(\log_2 N + T)$ time in total.

In order to externalize the interval tree structure in an efficient way, we need to increase the fan-out of the base tree to decrease its height to $O(\log_B N)$. This creates several problems. The main idea behind our successful externalization of the structure, as compared with previous attempts [16, 39], is to use a fan-out of $\sqrt{B}$ instead of B (following ideas from [7, 10]).

Structure. The external interval tree on a set of intervals I with endpoints in a fixed set E of size N is defined as follows. (We assume without loss of generality that the endpoints of the intervals in I are distinct.) The *base tree* $\mathcal{T}$ is a perfectly balanced fan-out $\sqrt{B}$ tree over the sorted set of endpoints E. Each leaf represents B consecutive points from E. (If $|E|$ is not $(\sqrt{B})^i B$ for some $i \geq 0$ we adjust the degree of the root of $\mathcal{T}$ to be smaller than $\sqrt{B}$.) The tree has height $O(\log_{\sqrt{B}}(N/B)) + 1 = O(\log_B N)$. As in the internal case, with each internal node v we associate an interval X_v consisting of all endpoints below v. The interval X_v is divided into $\sqrt{B}$ subintervals by the intervals associated with the children $v_1, v_2, \ldots, v_{\sqrt{B}}$ of v. Refer to Figure 2.1. For illustrative purposes, we call the subintervals *slabs* and the left (right) endpoint of a slab a *slab boundary*. We define a *multislab* to be a contiguous range of slabs, such as, for example, $X_{v_2} X_{v_3} X_{v_4}$ in Figure 2.1. In a node v we store intervals from I that cross one or more of the slab boundaries associated with v but none of the slab boundaries associated with *parent*(v). In a leaf l we store intervals with both endpoints among the endpoints in l. The number of intervals stored in a leaf is less than $B/2$ and can therefore be stored in one block. We store the set of intervals $I_v \subset I$ associated with v in the following $\Theta(B)$ secondary structures associated with v.

- For each of $\sqrt{B} - 1$ slab boundaries b_i, $1 < i \leq \sqrt{B}$, we store the following:
  - A right *slab list* R_i containing intervals from I_v with right endpoint between b_i and b_{i+1}. R_i is sorted according to right endpoints.
  - A left *slab list* L_i containing intervals from I_v with left endpoint between b_i and b_{i-1}. L_i is sorted according to left endpoints.
  - $O(\sqrt{B})$ *multislab lists*—one for each boundary to the right of b_i. The list $M_{i,j}$ for boundary b_j $(j > i)$ contains intervals from I_v with left endpoint between b_{i-1} and b_i and right endpoint between b_j and b_{j+1}. $M_{i,j}$ is sorted according to right endpoints.
- If the number of intervals stored in a multislab list $M_{i,j}$ is less than $\Theta(B)$, we instead store them in an *underflow structure* U along with intervals associated with all the other multislab lists with fewer than $\Theta(B)$ intervals. More precisely, only if more than B intervals are associated with a multislab do we store the intervals in the multislab list. Similarly, if fewer than $B/2$ intervals are associated with a multislab, we store the intervals in the underflow structure. If the number of intervals is between $B/2$ and B, they can be stored in either the multislab list or in the underflow structure. Since $O((\sqrt{B})^2) = O(B)$ multislabs lists are associated with v, the underflow structure U always contains fewer than B^2 intervals.

We implement all secondary list structures associated with v using B-trees and the underflow structure using a corner structure (Lemmas 2.1 and 2.3). In each node v, in $O(1)$ *index blocks*, we also maintain information about the size and place of each

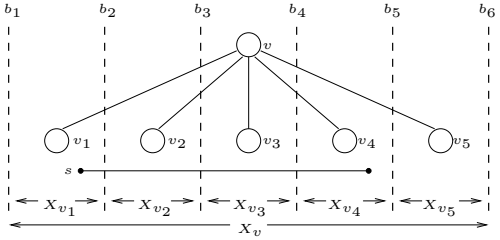


FIG. 2.1. *A node in the base tree. Interval s is stored in L_2, R_4, and either $M_{2,4}$ or U.*

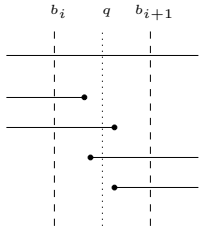


FIG. 2.2. *Intervals containing q are stored in R_{b_i}, $L_{b_{i+1}}$, the multislab lists spanning the slab, and U.*

of the $O(B)$ structures associated with v.

With the definitions above, an interval in I_v is stored in two or three structures: two slab lists L_i and R_j and possibly in either a multislab list $M_{i,j}$ or in the underflow structure U. For example, we store interval s in Figure 2.1 in the left slab list L_2 of b_2, in the right slab list R_4 of b_4, and in either the multislab list $M_{2,4}$ corresponding to b_2 and b_4 or the underflow structure U. Note the similarity between the slab lists and the two sorted lists of intervals in the nodes of an internal interval tree. As in the internal case, s is stored in a sorted list for each of its two endpoints. This represents the part of s to the left of the leftmost boundary contained in s and the part to the right of the rightmost boundary contained in s. Unlike in the internal case, in the external case we also need to represent the part of s between the two extreme boundaries. We do so using one of $O(B)$ multislab lists.

The external interval tree uses linear space: the base tree $\mathcal{T}$ itself uses $O(|E|/B)$ blocks, and each interval is stored in a constant number of linear space secondary structures (Lemmas 2.1 and 2.3). The number of other blocks used in a node is $O(\sqrt{B})$: $O(1)$ index blocks and one block for the underflow structure and for each of the $2\sqrt{B}$ slab lists. Since $\mathcal{T}$ has $O(|E|/(B\sqrt{B}))$ internal nodes, the structure uses a total of $O(|E|/B)$ blocks. Note that if we did not store the sparse multislab lists in the underflow structure, we could have $\Omega(B)$ sparsely utilized blocks in each node, which would result in a superlinear space bound.

Query. In order to answer a stabbing *query* q, we search down $\mathcal{T}$ for the leaf containing q, reporting all relevant intervals among the intervals I_v stored in each node v encountered. Assuming q lies between slab boundaries b_i and b_{i+1} in v, we report the relevant intervals in I_v as follows:

- We load the $O(1)$ index blocks.
- We report intervals in all multislab lists containing intervals crossing b_i and b_{i+1}, that is, multislab lists $M_{l,k}$ with $l \le i$ and $k > i$.
- We perform a stabbing query with q on the underflow structure U and report the result.
- We report intervals in R_i from the largest toward the smallest (according to right endpoint) until we encounter an interval not containing q.
- We report intervals in L_{i+1} from the smallest toward the largest until we encounter an interval not containing q.

It is easy to see that our algorithm reports all intervals in I_v containing q: all relevant intervals are stored either in a multislab list $M_{l,k}$ with $l \le i < k$, in U, in R_i, or in L_{i+1}. Refer to Figure 2.2. We correctly report all intervals in R_i containing q, since if an interval in the right-to-left order of this list does not contain q, then neither

does any other interval to the left of it. A similar argument holds for the left-to-right search in L_{i+1}.

The query algorithm uses an optimal $O(\log_B N + T/B)$ I/Os. In $\mathcal{T}$ we visit $O(\log_{\sqrt{B}} N) = O(\log_B N)$ nodes. In each node v we use only $O(1)$ I/Os that are not "paid for" by reportings (blocks read that contain $\Theta(B)$ output intervals): we use $O(1)$ I/Os to load the index blocks, $O(1)$ overhead to query U, and $O(1)$ overhead for R_i and L_{i+1}. Note how U is crucial for obtaining the $O(\log_B N + T/B)$ bound since it guarantees that all visited multislab lists contain $\Theta(B)$ intervals.

LEMMA 2.4. *Assuming $M \geq B^2$, there exists a data structure using $O(N/B)$ disk blocks to store intervals with unique endpoints in a set E of size N such that stabbing queries can be answered in $O(\log_B N + T/B)$ I/Os.*

Updates. We insert a new interval s in the external interval tree as follows: we search down $\mathcal{T}$ to find the first node v where s contains one or more slab boundaries. Then we load the $O(1)$ index blocks of v and insert s into the two relevant slab lists L_i and R_j. If the multislab list $M_{i,j}$ exists, we also insert s there. Otherwise, the other intervals (if any) corresponding to $M_{i,j}$ are stored in the underflow structure U, and we insert s in this structure. If that brings the number of intervals corresponding to $M_{i,j}$ up to B, we delete them all from U and insert them in $M_{i,j}$. Finally, we update and store the index blocks. Similarly, in order to delete an interval s, we search down $\mathcal{T}$ until we find the node storing s. We then delete s from two slab lists L_i and R_j. We also delete s from U or $M_{i,j}$; if s is deleted from $M_{i,j}$ and the list now contains $B/2$ intervals, we delete all intervals in $M_{i,j}$ and insert them into U. Finally, we again update and store the index blocks.

To analyze the number of I/Os used to perform an update, first note that for both insertions and deletions we use $O(\log_B N)$ I/Os to search down $\mathcal{T}$, and then in *one* node we use $O(\log_B N)$ I/Os to update the secondary list structures. The manipulation of the underflow structure U uses $O(1)$ I/Os, except in the cases where $\Theta(B)$ intervals are moved between U and a multislab list $M_{i,j}$. In the latter case we use $O(B)$ I/Os, but then there must have been at least $B/2$ updates involving intervals in $M_{i,j}$ and requiring only $O(1)$ I/Os since the last time an $O(B)$ cost was incurred. Hence the amortized I/O cost is $O(1)$, and we obtain the following.

THEOREM 2.5. *Assuming $M \geq B^2$, there exists a data structure using $O(N/B)$ disk blocks to store intervals with unique endpoints in a set E of size N such that stabbing queries can be answered in $O(\log_B N + T/B)$ I/Os in the worst case and such that updates can be performed in $O(\log_B N)$ I/Os amortized.*

3. General external interval tree. In order to remove the fixed endpoint assumption from our external interval tree, we need to use a dynamic search tree as the base tree. In internal memory a BB[α]-tree [34] is often used as the base tree for structures with secondary structures. In such a tree, a node v with weight w (i.e., with w elements below it) can be involved in a rebalancing operation only once for every $\Omega(w)$ updates that access (i.e., pass through) v [17, 33]. If the necessary reorganization of the secondary structures after a rebalance operation on v can be performed in $O(w)$ time, we then obtain an $O(1)$ amortized bound on performing a rebalancing operation. Unfortunately, a BB[α]-tree is not suitable for implementation in external memory; it is binary, and there seems to be no easy way of grouping nodes together in order to increase the fan-out while at the same time maintaining its other useful properties. On the other hand, a B-tree, which is the natural choice as dynamic base structure, does not have the property that a node v of weight w can be involved in a rebalance operation only for every $\Omega(w)$ updates accessing v.

In section 3.1, we describe a variant of B-trees, called *weight-balanced B-trees*, combining the useful properties of B-trees and BB[α]-trees. They are balanced using normal B-tree operations (split and fusion of nodes) while at the same time having the weight property of a BB[α]-tree. An important feature of a weight-balanced B-tree is that the ratio between the largest and smallest weight subtree rooted in children of a node v is a small constant factor. In a B-tree this ratio can be exponential in the height of the subtrees. In section 3.2, we use the weight-balanced B-tree to remove the fixed endpoint assumption from our external interval tree.

3.1. Weight-balanced B-tree. In a normal B-tree [12, 21] all leaves are on the same level, and each internal node has between a and $2a-1$ children for some constant a. In a weak B-tree, or (a,b)-tree [27], a wider range in the number of children is allowed. We define the weight-balanced B-tree by imposing constraints on the weight of subtrees rather than on the number of children. The other B-tree characteristics remain the same: the leaves are all on the same level (level 0), and rebalancing is performed by splitting and fusing internal nodes.

DEFINITION 3.1. *The weight $w(v_l)$ of a leaf v_l is defined as the number of elements stored in it. The weight of an internal node v is defined as $w(v) = \sum_{v=parent(c)} w(c)$.*

COROLLARY 3.2. *The weight $w(v)$ of an internal node v is equal to the number of elements in leaves below v.*

DEFINITION 3.3. *$\mathcal{T}$ is a weight-balanced* B-tree *with branching parameter a and leaf parameter k, $a > 4$ and $k > 0$, if the following conditions hold:*

- *All leaves of $\mathcal{T}$ are on the same level and have weight between k and $2k-1$.*
- *An internal node on level l has weight less than $2a^l k$.*
- *Except for the root, an internal node on level l has weight larger than $\frac{1}{2}a^l k$.*
- *The root has more than one child.*

LEMMA 3.4. *Except for the root, all nodes in a weight-balanced B-tree with parameters a and k have between $a/4$ and $4a$ children. The root has between 2 and $4a$ children.*

Proof. The leaves fulfill the internal node weight constraint, since $k > \frac{1}{2}a^0 k$ and $2k-1 < 2a^0 k$. Thus the minimal number of children an internal node on level l can have is $\frac{1}{2}a^l k / 2a^{l-1}k = a/4$, and the maximal number of children v can have is $2a^l k / \frac{1}{2}a^{l-1}k = 4a$. The root upper bound follows from the same argument, and the lower bound is by definition. ∎

COROLLARY 3.5. *The height of an N element weight-balanced B-tree with parameters a and k is $O(\log_a(N/k))$.*

To perform an *update* on a weight-balanced B-tree $\mathcal{T}$, we first search down $\mathcal{T}$ for the relevant leaf. After performing the actual update, we may need to rebalance $\mathcal{T}$ in order to fulfill the constraints in Definition 3.3. For simplicity we consider only insertions in this section. Deletions can easily be handled using global rebuilding [36] (as discussed further in the next section). After inserting an element in leaf u of $\mathcal{T}$, the nodes on the path from u to the root of $\mathcal{T}$ can be out of balance; that is, the node v_l on level l can have weight $2a^l k$. In order to rebalance the tree we split all such nodes starting with u and working towards the root. If u is a leaf containing $2k$ elements we split it into two leaves u and u', each containing k elements, and insert a reference to u' in $parent(u)$. In general, on level l we want to split a node v_l of weight $2a^l k$ into two nodes v_l' and v_l'' of weight $a^l k$ and insert a reference in $parent(v_l)$. (If $parent(v_l)$ does not exist, that is, if we are splitting the root, we create a new root with two children.) However, a perfect split is generally not possible if we want to perform the split so that v_l' gets the first (leftmost) i of v's children and v_l'' gets the rest of the

children. Nonetheless, since nodes on level $l-1$ have weight less than $2a^{l-1}k$, we can always find an i such that if we split at the ith child the weights of both v'_l and v''_l are between $a^lk - 2a^{l-1}k$ and $a^lk + 2a^{l-1}k$. Since $a > 4$, v'_l and v''_l fulfill the constraints of Definition 3.3; that is, their weights are strictly between $\frac{1}{2}a^lk$ and $2a^lk$.[3] Note that splitting node v does not change the weight of $parent(v)$. As a result, the structure is relatively simple to implement. In each node we need only to store its level and the weight of each of its children, information we can easily maintain during an update. The previous discussion and Corollary 3.5 combine to prove the following.

LEMMA 3.6. *The number of rebalancing operations (splits) after an insertion in a weight-balanced B-tree $\mathcal{T}$ with parameters a and k is bounded by $O(\log_a(|T|/k))$.*

The following lemma will be crucial in our application.

LEMMA 3.7. *After a split of a node v_l on level l into two nodes v'_l and v''_l, at least $a^lk/2$ inserts have to be performed below v'_l (or v''_l) before it splits again. After a new root r in a tree containing N elements is created, at least $3N$ insertions have to be performed before r splits again.*

Proof. After a split of v_l the weight of each of v'_l and v''_l is less than $a^lk + 2a^{l-1}k < 3/2a^lk$. Each such node will split again when its weight reaches $2a^lk$. It follows that the weight must increase by at least $a^lk/2$. When a root r is created on level l it has weight $2a^{l-1}k = N$. It will not split before it has weight $2a^lk > 2 \cdot 4a^{l-1}k = 4N$. □

One example of how the weight-balanced B-tree can be used as a simpler alternative to existing *internal memory* data structures is in adding range restriction capabilities to dynamic data structures [42]. The general technique for adding range restrictions developed in [42] utilizes a base BB[α]-tree with each interval node v augmented with a dynamic data structure on the set of elements below v. This structure needs to be rebuilt when a rebalancing operation is performed on v, and the use of a BB[α]-tree leads to amortized bounds. In [42] it is shown how worst-case bounds can be obtained by a relatively complicated redefinition of the BB[α]-tree. On the other hand, using our weight-balanced B-tree with branching parameter $a = 5$ and leaf parameter $k = 1$ as base tree we immediately obtain worst-case bounds: the large number of updates between splits of a node immediately implies good amortized bounds, and the bounds can easily be made worst-case by performing the secondary structure rebuilding lazily. The ideas and techniques used in this construction are very similar to the ones presented in the succeeding sections of this paper and are therefore omitted. The use of lazy rebuilding in the BB[α]-tree solution is complicated because rebalancing is performed using rotations, which means that we cannot simply continue to query and update the old secondary structure while lazily building new ones.

As mentioned, the weight-balanced B-tree has been used in the development of numerous efficient internal as well as external data structures (e.g., [26, 13, 8, 25, 9, 2, 41]). In order to obtain an external tree structure suitable for use in our interval tree, we choose $4a = \sqrt{B}$ and $2k = B$ and obtain the following.

THEOREM 3.8. *There exists an N element search tree data structure using $O(N/B)$ disk blocks such that a search or an insertion can be performed in $O(\log_B N)$ I/Os in the worst case.*

Each internal node v at level l in the structure, except for the root, has $\Theta(\sqrt{B})$ children, dividing the $w(v) = \Theta((\sqrt{B})^lB)$ elements below v into $\Theta(\sqrt{B})$ sets. The root

[3] If $a > 8$ we can even split the node at child $i-1$ or $i+1$ instead of at child i and still fulfill the constraints. We will use this property in the next section.

has $O(\sqrt{B})$ children. Rebalancing after an insertion is performed by splitting nodes. When a node v is split, at least $\Theta(w(v))$ elements must have been inserted below v since the last time v was split. In order for a new root to be created, $\Theta(N)$ elements have to be inserted into the data structure.

Proof. Each internal node can be represented using $O(1)$ blocks, and the space bound follows since each leaf contains $\Theta(B)$ elements. A split at v can be performed in $O(1)$ I/Os: we load the $O(1)$ blocks storing v into internal memory, split v, and write the $O(1)$ blocks defining the two new nodes back to disk. Finally, we update the information in the parent using $O(1)$ I/Os. Thus the insertion and search I/O bounds follow directly from Corollary 3.5 and Lemma 3.6.

The second part of the theorem follows directly from Definition 3.3, Corollary 3.2, and Lemmas 3.4 and 3.7. □

3.2. Using the weight-balanced B-tree to remove the fixed endpoint assumption. We now show how to remove the fixed endpoint assumption from our external interval tree using the weight-balanced B-tree as the base tree $\mathcal{T}$. To insert an interval, we first insert the two new endpoints in the base tree and perform the necessary rebalancing. Then we insert the interval as described in section 2. Since rebalancing is performed by splitting nodes, we need to consider how to split a node v in our interval tree. Figure 3.1 illustrates how the slabs associated with v are affected when v splits into nodes v' and v'': All the slabs on one side of a slab boundary b get associated with v'; the boundaries on the other side of b get associated with v''; and b becomes a new slab boundary in *parent*(v). As a result, all intervals in the secondary structures of v that contain b need to be inserted into the secondary structures of *parent*(v). The rest of the intervals need to be stored in the secondary structures of v' and v''. Furthermore, as a result of the addition of the new boundary b, some of the intervals in *parent*(v) containing b also need to be moved to new secondary structures. Refer to Figure 3.2.

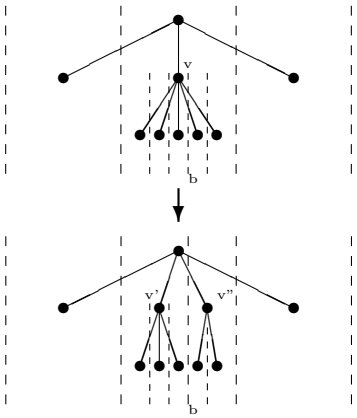


FIG. 3.1. *Splitting a node; v splits along b, which becomes a new boundary in parent(v).*

First consider the intervals in the secondary structures of v. Since each interval is stored in a left slab list and a right slab list, we can collect all intervals containing b (to be moved to *parent*(v)) by scanning through all of v's slab lists. We first construct a list L_r of the relevant intervals sorted according to right endpoint by scanning through the right slab lists. We scan through every right slab list (stored in the leaves of a B-tree) of v in order, starting with the rightmost slab boundary,

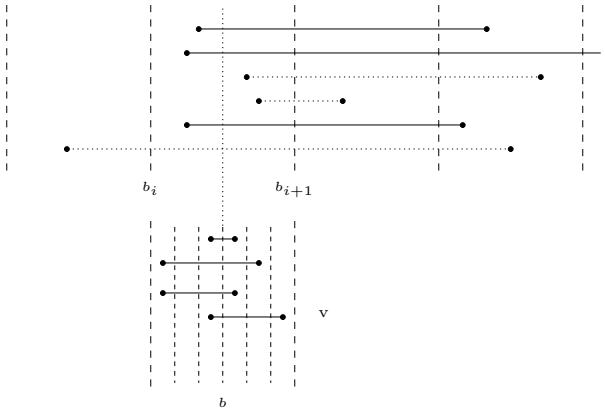


FIG. 3.2. *All solid intervals need to move. Intervals in v containing b move to parent(v), and some intervals move within parent(v).*

adding intervals containing b to L_r. This way L_r will automatically be sorted. We construct a list L_l sorted according to left endpoint by scanning through the left slab lists in a similar way. Since the secondary structures of v contain $O(w(v))$ intervals (they all have an endpoint below v), and since we can scan through each of the $O(\sqrt{B})$ slab lists in a linear number of I/Os (Lemma 2.1), we construct L_r and L_l in $O(\sqrt{B} + w(v)/B) = O(w(v)/B)$ I/Os. Next we construct the slab lists of v' and v'', simply by removing intervals containing b from each slab list of v. We remove the relevant intervals from a given slab list by scanning through the leaves of its B-tree, collecting the intervals for the new list in sorted order, and then constructing a new list (B-tree). This way we construct all the slab lists in $O(w(v)/B)$ I/Os. We construct the multislab lists for v' and v'' simply by removing all multislabs lists containing b. Since each removed list contains $\Omega(B)$ intervals, we can do so in $O(w(v)/B)$ I/Os. We construct the underflow structures for v' and v'' by first scanning through the underflow structure for v and collecting the intervals for the two structures, and then constructing them individually using $O(w(v)/B)$ I/Os (Lemma 2.3). We complete the construction of v' and v'' in $O(w(v)/B)$ I/Os by scanning though the lists of each of the nodes, collecting the information for the index blocks.

Next consider *parent*(v). We need to insert the intervals in L_l and L_r into the secondary structures of *parent*(v) and move some of the intervals already in these structures. The intervals we need to consider all have one of their endpoints in X_v. For simplicity we consider only intervals with *left* endpoint in X_v; intervals with right endpoint in X_v are handled similarly. All intervals with left endpoint in X_v that are stored in *parent*(v) cross boundary b_{i+1}. Thus we need to consider each of these intervals in one or two of $\sqrt{B} + 1$ lists, namely, in the left slab list L_{i+1} of b_{i+1} and possibly in one of $O(\sqrt{B})$ multislab lists $M_{i+1,j}$. When introducing the new slab boundary b, some of the intervals in L_{i+1} need to be moved to the new left slab list of b. In a scan through L_{i+1} we collect these intervals in sorted order in $O(|X_v|/B) = O(w(v)/B)$ I/Os. The intervals in L_l also need to be stored in the left slab list of b, so we merge L_l with the collected list of intervals and construct a B-tree on the resulting list. We can easily do so in $O(w(v)/B)$ I/Os (Lemma 2.1), and we can update L_{i+1} in the same bound. Similarly, some of the intervals in multislab lists $M_{i+1,j}$ need to be moved to new multislab lists corresponding to multislabs with

b as left boundary instead of b_{i+1}. We can easily move the relevant intervals (and thus construct the new multislab lists) in $O(w(v)/B)$ I/Os using a scan through the relevant multislab lists, similarly to the way we moved intervals from the left slab list of b_{i+1} to the left slab list of b. (Note that intervals in the underflow structure do not need to be moved.) If any of the new multislab lists contain fewer than $B/2$ intervals, we instead insert the intervals into the underflow structure U. We can easily do so in $O(B) = O(w(v)/\sqrt{B})$ I/Os by rebuilding U. Finally, to complete the split process we update the index blocks of $parent(v)$.

To summarize, we can split a node v in $O(w(v)/\sqrt{B})$ I/Os, and since $O(w(v))$ endpoints must have been inserted below v since it was constructed (Theorem 3.8), the amortized cost of a split is $O(1/\sqrt{B})$ I/Os. Since $O(\log_B N)$ nodes split during an insertion, we obtain the following.

LEMMA 3.9. *Assuming $M \geq B^2$, there exists a data structure using $O(N/B)$ disk blocks to store N intervals such that stabbing queries can be answered in $O(\log_B N + T/B)$ I/Os in the worst case and such that an interval can be inserted in $O(\log_B N)$ I/Os amortized.*

As mentioned in section 3.1, deletions can be handled using global rebuilding [36]. To delete an interval s we first delete it from the secondary structures as described in section 2 *without* deleting the endpoints of s from the base tree $\mathcal{T}$. Instead we just mark the two endpoints in the leaves of the base tree as deleted. This does not increase the number of I/Os needed to perform a later update or query operation, but it does not decrease it either. After $N/2$ deletions have been performed we rebuild the structure in $O(N \log_B N)$ I/Os, leading to an $O(\log_B N)$ amortized delete I/O bound: first we scan through the leaves of the old base tree and construct a sorted list of the undeleted endpoints. This list is then used to construct the new base tree. All of this can be done in $O(N/B)$ I/Os. Finally, we insert the $O(N)$ intervals one by one without rebalancing the base tree, using $O(N) \cdot O(\log_B N)$ I/Os.

THEOREM 3.10. *Assuming $M \geq B^2$, there exists an external interval tree using $O(N/B)$ disk blocks to store N intervals such that stabbing queries can be answered in $O(\log_B N + T/B)$ I/Os in the worst case and such that updates can be performed in $O(\log_B N)$ I/Os amortized.*

4. Removing amortization. In this section, we discuss how to make the update bound of Theorem 3.10 worst-case. Amortized bounds appeared in several places in our structure; in the fixed endpoint version of our structure, amortization was introduced as a result of the amortized $O(1)$ update bound of the underflow structure (Lemma 2.3), as well as when moving intervals between the underflow and multislab lists. In the dynamic base tree version, amortization was introduced in the amortized node split bound (insertions) as well as in the use of global rebuilding (deletions).

In sections 4.1 and 4.2, we show how to remove amortization from the node split and underflow (corner structure) update bounds, respectively. In section 4.2, we also show how to remove the $M \geq B^2$ assumption from the corner structure and thus from our external interval tree. The remaining amortization can be removed using standard lazy global rebuilding techniques [36]. We make the global rebuilding (deletion) bound worst-case as follows: instead of using $O(N \log_B N)$ I/Os to rebuild the entire structure when the number of endpoints fall below $N/2$, we distribute the rebuilding over the next $1/3 \cdot N/2$ updates using $O(\log_B N)$ I/Os on rebuilding at each update. We use and update the original structure while constructing the new structure. When the new structure is completed the $1/3 \cdot N/2$ updates that occurred after the rebuilding started still need to be performed in the new structure. We

perform these updates during the next $1/3 \cdot (1/3 \cdot N/2)$ operations. This process continues until both structures store the same set of intervals (with at least $(1 - (1/3 + 1/9 + \cdots))N/2 \geq 1/2 \cdot N/2$ endpoints). Then we dismiss the old structure and use the new one instead. Since the rebuilding is finished before the structure contains only $N/4$ endpoints, we are in the process of constructing at most one new structure at any given time. The amortization introduced when moving intervals between the underflow structure and multislab lists can be removed in a similar way: recall that we moved B intervals using $O(B)$ I/Os when the underflow structure contained B intervals belonging to the same multislab list $M_{i,j}$ and when the number of intervals in a multislab list fell below $B/2$. We remove this amortization by moving the intervals over $B/4$ updates. If the size of a multislab list $M_{i,j}$ falls to $B/2$, we move two intervals from $M_{i,j}$ to the underflow structure over each of the next $B/4$ insertions or deletions involving $M_{i,j}$. Insertions themselves are also performed on the underflow structure, and deletions are performed on the underflow structure or $M_{i,j}$. When all intervals are moved there are between $\frac{1}{4}B$ and $\frac{3}{4}B$ intervals belonging to $M_{i,j}$ stored in U. Similarly, if the number of intervals in the underflow structure belonging to $M_{i,j}$ reaches B, we move the B intervals during the next $B/4$ updates involving $M_{i,j}$. Even though this way $M_{i,j}$ can contain $o(B)$ intervals, the optimal space and query bounds are maintained since $M_{i,j}$ and U together contain $\Theta(B)$ intervals during the process. This proves our main result.

THEOREM 4.1. *There exists an external interval tree using $O(N/B)$ disk blocks to store N intervals such that stabbing queries can be answered in $O(\log_B N + T/B)$ I/Os in the worst case and such that updates can be performed in $O(\log_B N)$ I/Os in the worst case.*

4.1. Splitting nodes lazily. Recall that when a node v splits along a boundary b the intervals in v containing b need to be moved to *parent*(v), and some of the intervals in *parent*(v) containing b need to move internally in *parent*(v) (Figure 3.2). In section 3, we showed how to move the intervals in $O(w(v)/\sqrt{B})$ I/Os, and since $O(w(v))$ updates have to be performed below v between splits (Theorem 3.8) we obtained an $O(1/\sqrt{B})$ amortized split bound. When performing an update in a leaf l it affects the weight of $O(\log_B N)$ nodes on the path from the root to l. These nodes are all *accessed* in the search for l performed before the actual insertion. In this section we show how to split a node v (move the relevant intervals) lazily using $O(1)$ I/Os during the next $O(w(v))$ updates accessing v while still being able to query the secondary structures of v efficiently. This way we are done splitting v before a new split is needed, and we obtain an $O(1)$ worst-case split bound.

Our lazy node splitting algorithm works as follows. When v needs to be split along a boundary b, in $O(1)$ I/Os we first insert b as a *partial* slab boundary into *parent*(v). The boundary remains partial until we have finished the split. In order to keep different split processes from interfering with each other, we want to avoid splitting nodes along partial boundaries. Since v, or rather the nodes it splits into, cannot split again as long as b is partial, at most every second boundary in *parent*(v) can be partial. As discussed in section 3.1 (footnote 3), this means that we can always split a node along a nonpartial boundary without violating the constraints on the base weight-balanced B-tree $\mathcal{T}$. Next we move the relevant intervals in two phases: in an *up phase*, we first construct the new secondary structures for v' and v'' as before, by removing the intervals in secondary structures of v containing b and collecting them in two sorted lists L_l and L_r. Below we show how to do so lazily using $O(1)$ I/Os over $O(w(v)/\sqrt{B})$ updates accessing v so that we can still

query and update the "old" secondary structures of v. During this phase we can therefore perform queries as before, simply by ignoring the partial slab boundary b in *parent*(v). Next, in the *rearrange phase*, in $O(1)$ I/Os, we split v into v' and v'' by switching to the new structures and splitting the index blocks. We also associate the lists L_l and L_r with the partial boundary b in *parent*(v). Then we move the relevant intervals in *parent*(v) containing b by constructing new updated versions of all secondary structures containing intervals with endpoints in X_v. Below we discuss how to do so lazily over $O(w(v)/\sqrt{B})$ updates accessing v (v' and v'') so that the "old" structures can still be queried and updated. In order to answer queries between b_i and b_{i+1} in *parent*(v) correctly during the rearrange phase, we first perform a query as before while ignoring the partial slab boundary b. To report the relevant intervals among the intervals we have removed from v and inserted into L_l and L_r, we then query the relevant one of the two slab lists. Since this adds only $O(1)$ I/Os to the query procedure in *parent*(v), the optimal query bound is maintained. At the end of the rearrange phase, we finish the split of v in $O(1)$ I/Os by switching to the new structures and marking b as a normal slab boundary.

All that remains is to describe how to perform the up and rearrange phases. Each of these phases can be performed lazily using $O(1)$ I/Os during each of $O(w(v)/\sqrt{B})$ updates accessing v. However, our algorithms will assume that only one up or rearrange phase is in progress on a node v at any given time, which means that when we want to perform a phase on v we might need to wait until we have finished another phase. In fact, other phases may also be waiting, and we may need to wait until v has been accessed enough times for us to finish all of them. Luckily, since an up and rearrange phase requires only $O(w(v)/\sqrt{B})$ accesses to finish, and since we need only to finish our phase in $O(w(v))$ accesses, we can afford to wait for quite a while. Before an up phase can be performed on v we at most have to finish $O(\sqrt{B})$ rearrange phases (one for each partial slab boundary), each requiring $O((w(v)/\sqrt{B})/\sqrt{B}) = O(w(v)/B)$ accesses, for a total of $O(w(v)/\sqrt{B})$ accesses. After finishing the up phase on v we might need to finish $O(\sqrt{B})$ other rearrange phases and one up phase on *parent*(v) before performing the rearrange phase. These phases require at most $O(\sqrt{B}) \cdot O(w(v)/\sqrt{B}) + O((w(v) \cdot \sqrt{B})/\sqrt{B}) = O(w(v))$ accesses. Thus in total we finish the split of v in the allowed $O(w(v))$ accesses. Note that the waiting can be implemented simply by associating a single block with v storing a queue with information about the $O(\sqrt{B})$ phases waiting at v. When we want to perform a phase on v we simply insert it in the queue, and each time v is accessed $O(1)$ I/Os are performed on the phase in the front of the queue. When v has been accessed enough times to finish one phase we start the next phase in the queue. Note also that while we are waiting to perform a phase on a node v, or even while we are performing the phase, new partial slab boundaries may be inserted in v, and new slab lists may be inserted for such boundaries (due to splits of children of v). However, this does not interfere with the up or rearrange phase, since we do not split along partial boundaries and since the intervals in the two new slab lists for a partial boundary contain only the partial boundary.

After having described how to guarantee that we are working on only one up or rearrange phase in a node v at any given time, we can now describe how to perform such a phase lazily over $O(w(v)/\sqrt{B})$ accesses. We do so basically using lazy global rebuilding: during normal updates we maintain a copy of each secondary structure—called a *shadow* structure. Maintaining such shadow structures along with the original structures does not change the asymptotic space, update, or query bounds of the

interval tree. When we start an up or rearrange phase on v, we first "freeze" the shadow structures of v; that is, instead of performing updates on them we just store updates as they arrive. Then we perform the necessary movement of intervals on the *shadow* structures as described in section 3. It is easy to realize how we can perform these movements over the next $O(w(v)/\sqrt{B})$ updates accessing v such that $O(1)$ I/Os are used at each access. The only slight complication is that we need to make sure that updates performed in v during the process (that is, updates that were stored at v and still need to be performed on the shadow structures) are performed after we have moved the relevant intervals. To do so within the $O(w(v)/\sqrt{B})$ bound, we actually perform $\Theta(\log_B N)$ instead of only $O(1)$ I/Os on the shadow structure movement process every time an update is performed in v. This does not change the overall $O(\log_B N)$ update I/O bound, since an update (insertion or deletion of an interval) takes place only in *one* node of $\mathcal{T}$ (or, equivalently, since we are already using $O(\log_B N)$ I/Os to perform the update on the original secondary structures of v). After finishing the interval movement, we then perform the stored updates lazily using $O(1)$ I/Os each time v is accessed. Since we performed $\Theta(\log_B N)$ I/Os on the shadow structures each time an update was stored at v, we are guaranteed that we will finish performing the updates within $O(w(v)/\sqrt{B})$ accesses to v. Finally, after moving intervals and performing updates, we (lazily) make a copy (shadow) of the new shadow structures in the same I/O bound. We handle updates performed during this copying in the same way we handled updates during global rebuilding of the external interval tree (to remove delete amortization). We finish the phase by discarding the old secondary structures and instead use the updated shadow structures (along with their shadow) as the secondary structures.

4.2. Removing amortization from the corner structure. In this section we sketch the "corner structure" of Kannelakis et al. [30] (Lemma 2.3) and discuss how its $O(1)$ amortized update bound can be made worst-case. At the same time we remove the assumption that $M \geq B^2$.

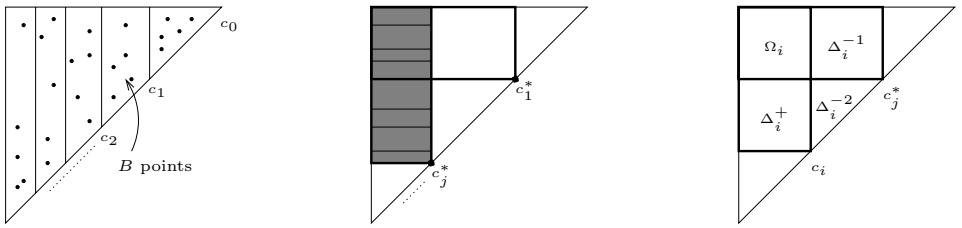


FIG. 4.1. *(a) Vertical regions. (b) The set C^* (the marked points). The dark lines represent the boundaries of queries whose corners are points in C^*. One (horizontally blocked) query is shaded. (c) The sets Ω_i, Δ_i^{-1}, Δ_i^{-2}, and Δ_i^+. c_j^* is the last point that was added to C^*, and c_i is being considered for inclusion in C^*.*

The corner structure is designed to store a set S of $K \leq B^2$ points in the plane above the $x = y$ line such that diagonal corner queries can be answered in $O(1+T/B)$ I/Os [30]. As discussed in the introduction, this problem is equivalent to the stabbing query problem. The structure is defined as follows: First S is divided into $\lceil K/B \rceil$ *vertical* regions containing B points each. Refer to Figure 4.1(a). The points in these regions are stored in $\lceil K/B \rceil$ blocks. Let C be the set of points at which the right boundaries of the $\lceil K/B \rceil$ regions intersect the $y = x$ line. An iterative procedure is

used to choose a subset C^* of these points, and one or more *horizontally* oriented blocks are used to explicitly store the answer to each query with a corner $c^* \in C^*$. Refer to Figure 4.1(b). First the highest point $c_1^* \in C$ is included in C^*. Then each point in C is considered in turn along the $x = y$ line. Let c_j^* be the point of C most recently added to C^*; initially, this is c_1^*. When considering $c_i \in C$, the sets $\Omega_i \subset S$, $\Delta_i^{-1} \subset S$, $\Delta_i^{-2} \subset S$, and $\Delta_i^+ \subset S$ are defined as shown in Figure 4.1(c). The set $S_j^* = \Omega_i \cup \Delta_i^{-1}$ is the answer to a query whose corner is c_j^*, and $S_i = \Omega_i \cup \Delta_i^+$ is the answer to a query whose corner is c_i. Let $\Delta_i^- = \Delta_i^{-1} \cup \Delta_i^{-2}$. The point c_i is then added to C^* if and only if

$$|\Delta_i^-| + |\Delta_i^+| > |S_i|.$$

In [30] it is shown that the total number of blocks used to store the sets S_i^* is $O(K/B)$. The corner structure consists of these blocks, as well as $O(1)$ blocks storing the sets C and C^*. In [30] it is also shown how a diagonal corner query can be answered in $O(1 + T/B)$ I/Os using this representation.

The corner structure can easily be constructed in $O(K/B)$ I/Os when $M \geq B^2$ (since in this case it fits in main memory). As discussed, the structure can therefore easily be made dynamic with an $O(1)$ amortized update bound using an update block and global rebuilding [30]. In the following we show how to construct the structure in $O(K/B)$ I/Os incrementally over $O(K/B)$ updates such that no more than $O(1)$ blocks are loaded into main memory at any time. This immediately removes the $M \geq B^2$ assumption. Using this result and lazy global rebuilding, the $O(1)$ amortized update bound can then be made worst-case: once $B/2$ updates have been collected in the update block, the structure is rebuilt during the next $B/2$ updates using $O(1)$ I/Os at each update.

We first discuss how to construct a corner structure on K points in $O(K/B)$ I/Os using $O(1)$ blocks of main memory. After that we discuss how the construction can be performed lazily. We assume that the K points are given in two lists sorted according to x- and y-coordinates, respectively. We can easily store two such lists along with the corner structure itself using $O(K/B)$ space. At the start of a rebuilding process, we first merge the $O(B)$ points in the update block into these two lists in $O(K/B)$ I/Os using $O(1)$ blocks of main memory. Then we construct the corner structure in three steps: first we compute the vertical blocking, that is, the set C. Then we compute C^*. Finally, we construct the horizontal blocking corresponding to each of the points in C^*.

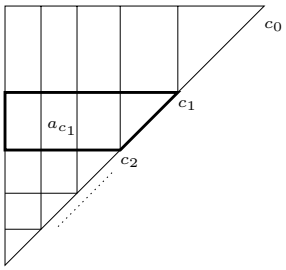
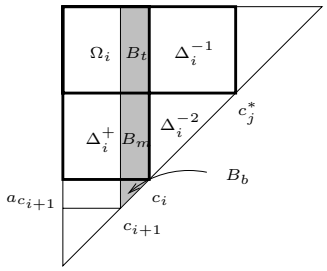


FIG. 4.2. *Definition of* a_{c_i}.

The vertical blocking is simply the list sorted according to x-coordinates. We can easily make a copy of this list and thus compute the set C in $O(K/B)$ I/Os



$$
\begin{aligned}
|\Delta_{i+1}^{-1}| &= |\Delta_i^{-1}| + |B_t| \\
|\Delta_{i+1}^{-2}| &= |\Delta_i^{-2}| + |B_m| + |B_b| \\
|\Omega_{i+1}| &= |\Omega_i| - |B_t| \\
|\Delta_{i+1}^{+}| &= |\Delta_i^{+}| - |B_m| + (a_{c_{i+1}} - |B_b|)
\end{aligned}
$$

FIG. 4.3. *Definition of B_t, B_m, and b_b and computation of C^*.*

using $O(1)$ blocks of memory. Let $c_i \in C$ be the ith point in the sorted sequence of points on the $x = y$ line. To aid the computation of C^* we first compute for each $c_i \in C$ a number a_{c_i}. We define a_{c_i} to be the number of points with y-coordinates between the y-coordinates of c_i and c_{i+1}. Refer to Figure 4.2. We compute all a_{c_i}s in $O(K/B)$ I/Os using $O(1)$ blocks of main memory in a single scan of the list of points sorted by y-coordinates. We then compute C^* as previously by proceeding along the $x = y$ line and including $c_i \in C$ in C^* if $|\Delta_i^-| + |\Delta_i^+| > |S_i|$. Since the number of points in C is $O(K/B)$, our goal is to decide if c_i should be included in $O(1)$ I/Os. We can do so if we can compute $|\Omega_{i+1}|$, $|\Delta_{i+1}^+|$, $|\Delta_{i+1}^{-1}|$, and $|\Delta_{i+1}^{-2}|$ in $O(1)$ I/Os, given $|\Omega_i|$, $|\Delta_i^+|$, $|\Delta_i^{-1}|$, $|\Delta_i^{-2}|$, c_j^*, c_i, c_{i+1}, and a_{i+1}. (These values/points can all be loaded into main memory in $O(1)$ I/Os.) Figure 4.3 illustrates how we can compute $|\Omega_{i+1}|$, $|\Delta_{i+1}^+|$, $|\Delta_{i+1}^{-1}|$, and $|\Delta_{i+1}^{-2}|$ once B_t, B_m, and B_b have been computed. We compute these three sets, containing a total of B points, in $O(1)$ I/Os, simply by loading the relevant vertical block. Thus overall we can compute C^* in $O(K/B)$ I/Os.

To compute and horizontally block the points in the answer S_j^* to a query at each of the points $c_j^* \in S^*$, we again proceed along the $x = y$ line considering each point $c_i^* \in C^*$ in turn. Assume we have already blocked S_j^* and that we know the position p of the last (lowest y-coordinate) point in S_j^* in the list of points sorted by y-coordinates. Initially, S_j^* is empty, and p is the first point in the list of points sorted by y-coordinates. To block S_{j+1}^*, we scan through the horizontal blocking of S_j^*, collecting the points with x-coordinate smaller than the x-coordinate of c_{j+1}^*. This way we obtain the points in S_{j+1}^* above the y-coordinate of s_j^* horizontally blocked (sorted by y-coordinates). Then we collect the remaining points in S_{j+1}^* horizontally blocked by scanning through the list of points sorted by y-coordinates, starting at p. Altogether we use $O(|S_j^*|/B + |S_{j+1}^*|/B)$ I/Os to compute the blocking of S_{j+1}^*. Thus we use $O(2\sum_{j \in 1..|S^*|} |S_j^*/B|)$ I/Os to compute the blocking corresponding to all the points in C^*. This is $O(K/B)$ since the structure uses linear space.

We have shown how to construct the corner structure in $O(K/B)$ I/Os using $O(1)$ blocks of main memory. We can easily modify the algorithm to work in an incremental way, that is, to run in $O(K/B)$ steps of $O(1)$ I/Os without using any main memory between two steps: throughout the algorithm we can represent the current state of the algorithm by a constant number of pointers and values. We can therefore perform one step by loading the current state into main memory using $O(1)$ I/Os, performing the step using $O(1)$ I/Os, and finally using $O(1)$ I/Os to write the new state back to disk. In total we have proved the following.

LEMMA 4.2. *A set of $K \leq B^2$ intervals can be stored in an external data structure using $O(K/B)$ disk blocks such that a stabbing query can be answered in $O(T/B+1)$ I/Os and such that updates can be performed in $O(1)$ I/Os. The structure can be constructed in $O(K/B)$ I/Os.*

5. External segment tree. In this section we sketch how the ideas used in the external interval tree can also be used to develop an external segment tree-like structure with a better space bound than previously known for such structures [15, 39]. Like an interval tree, a segment tree solves the stabbing query problem. Unlike the interval tree, however, it uses superlinear space. It is often used as base tree structure in multidimensional structures (refer, e.g., to [10, 3]).

Structure. In internal memory, as in the case of the interval tree, a segment tree consists of a binary base tree with intervals stored in secondary structures of internal nodes. Unlike for the interval tree, an interval can be stored in the secondary structures of up to two nodes on each level of the base tree. More precisely, an interval s is stored in all nodes v such that s contains the interval associated with at least one of v's children but not the interval X_v associated with v. As in the interval tree case, we externalize the structure by using a weight-balanced B-tree (Theorem 3.8) as the base tree. As previously, an internal node v defines $\Theta(\sqrt{B})$ slabs and $\Theta(B)$ multislabs, and $\Theta(B)$ secondary structures are associated with v. An interval s is stored only in the secondary structures of v if it spans one of v's *slabs* but not the whole interval X_v. Thus, unlike in the external interval tree, where s is stored only in the highest node for which it contains a slab *boundary*, s can be stored in $O(\log_B N)$ nodes—refer to Figure 5.1(a). As in the interval tree, s is stored in a multislab list corresponding to the largest multislab it spans, and as before intervals from multislab lists containing $o(B)$ intervals are stored in an underflow structure. Note how an external segment tree corresponds to an external interval tree, where parts of an interval not completely spanning a slab in a node v are stored recursively instead of in a slab list. Multislab lists are implemented as simple (unordered) lists, and with each interval s we store pointers to the copies of s in the nearest ancestor and descendent of v storing copies of s. These pointers are not directly maintained for intervals in underflow structures. Instead we keep a separate lists of intervals in the underflow structure of each node v, and pointers are maintained for these intervals. This allows us to rebuild an underflow structure containing K intervals in $O(K/B)$ I/Os and thus maintain the $O(1)$ update bound of the underflow structure (Lemma 4.2); maintaining pointers would have required $O(K)$ I/Os. Finally, we maintain an auxiliary B-tree containing all intervals in the structure, ordered according to right endpoint, such that given an interval s we can obtain a pointer to the copy of s stored in the topmost node storing s. This B-tree uses linear space, and since the secondary structures of a node also use linear space the external segment tree uses $O((N/B)\log_B N)$ disk blocks to store N intervals.

Query. We answer a stabbing *query* q on an external segment tree in $O(\log_B N + T/B)$ I/Os, simply by searching down the base tree for q and in each node querying the underflow structure and reporting all intervals in lists corresponding to multislabs containing q.

Updates. To *insert* an interval s into an external segment tree we first insert the endpoints of s in the base tree and rebalance the tree by splitting nodes. Below we show how a node v can be split in $O(w(v))$ I/O such that an endpoint is inserted in $O(\log_B N)$ I/Os amortized. Then we perform the actual insertion of s by traversing two paths in the base tree, inserting s in the relevant multislab lists. As the multislab lists are unsorted, we can insert s in a list in $O(1)$ I/O while at the same time

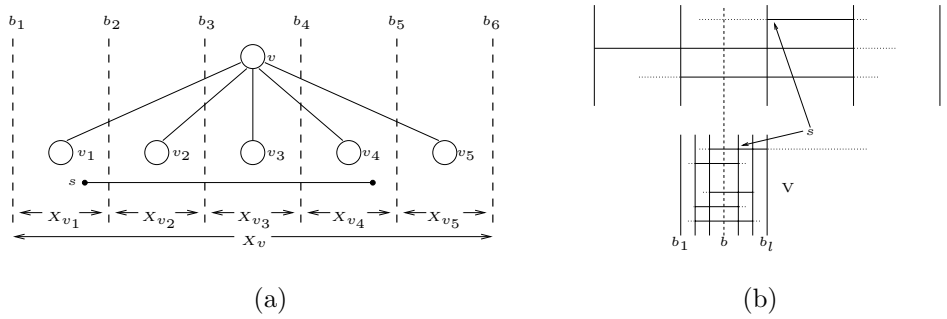


FIG. 5.1. (a) *Node v in the base tree. The interval s is stored in v as well as (recursively) in v_1 and v_4.* (b) *Splitting a segment tree node.*

storing the relevant pointers to other copies of s. We handle the cases involving the underflow structure in $O(1)$ I/Os precisely as in the external interval tree case. In total we can perform an insertion in $O(\log_B N)$ I/Os. We *delete* an interval s from an external segment tree in $O(\log_B N)$ I/Os simply by locating s in the auxiliary B-tree—obtaining a pointer to the topmost of the $O(\log_B N)$ occurrences of s—deleting s from the B-tree, and then using the pointers between copies of s to find and remove all occurrences of s in $O(\log_B N)$ I/Os; note that even though in a node where s is stored in the underflow structure we obtain a pointer to the separate list of intervals, we can still delete s from the underflow structure in $O(1)$ I/Os. As in the interval tree case, we remove the endpoints of s from the base tree in $O(\log_B N)$ I/Os using global rebuilding.

All that remains is to describe how to efficiently split a node v along a slab boundary b. When v splits into v' and v'', all intervals in multislabs containing b need to be moved. These intervals fall into two categories: intervals that contain the leftmost or rightmost slab boundaries b_1 and b_l of v and those that do not. Refer to Figure 5.1(b). Intervals not containing b_1 or b_l (but containing b) need to be stored in multislab lists of both v' and v''. Thus to move these intervals we simply need to make a copy of the relevant lists for both v' and v''. We also need to update the relevant pointers between intervals. We can easily do so in $O(w(v))$ I/Os. Intervals that contain b_1 or b_l (and b) need to be inserted in v' or v'', as well as moved within *parent*(v). Consider, for example, intervals containing b and b_l (e.g., interval s in Figure 5.1(b). Such intervals need to be inserted in multislab lists for v', as well as either inserted into *parent*(v) or moved within *parent*(v): if one of these intervals s is already stored in *parent*(v) we use the pointer stored with s in v to locate and delete s in *parent*(v), and then we insert s in the relevant new multislab list. Since each interval can be moved in $O(1)$ I/Os, and since all moved intervals have an endpoint in X_v, we use $O(w(v))$ I/Os in total to split a node v. Finally, as in the interval tree case, we can perform the split over $O(w(v))$ updates accessing v and thus obtain worst-case bounds.

THEOREM 5.1. *There exists an external segment tree using $O((N/B)\log_B N)$ disk blocks to store N intervals such that stabbing queries can be answered in $O(\log_B N + T/B)$ I/Os and such that updates can be performed in $O(\log_B N)$ I/Os.*

Acknowledgment. We thank two anonymous reviewers for comments that improved the presentation of the results in this paper.

REFERENCES

[1] P. K. Agarwal, L. Arge, G. S. Brodal, and J. S. Vitter, *I/O-efficient dynamic point location in monotone planar subdivisions*, in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 1999, pp. 11–20.

[2] P. K. Agarwal, L. Arge, and J. Erickson, *Indexing moving points*, J. of Comput. System Sci., 66 (2003), pp. 207–243.

[3] P. K. Agarwal and J. Erickson, *Geometric range searching and its relatives*, in Advances in Discrete and Computational Geometry, B. Chazelle, J. E. Goodman, and R. Pollack, eds., Contemp. Math. 223, AMS, Providence, RI, 1999, pp. 1–56.

[4] A. Aggarwal and J. S. Vitter, *The input/output complexity of sorting and related problems*, Comm. ACM, 31 (1988), pp. 1116–1127.

[5] L. Arge, *External memory data structures (invited paper)*, in Proceedings of the 9th Annual European Symposium on Algorithms, Aarhus, Denmark, 2001, Lecture Notes in Comput. Sci. 2161, Springer-Verlag, Berlin. 2001, pp. 1–29.

[6] L. Arge, *External memory data structures*, in Handbook of Massive Data Sets, J. Abello, P. M. Pardalos, and M. G. C. Resende, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002, pp. 313–358.

[7] L. Arge, *The buffer tree: A technique for designing batched external data structures*, Algorithmica, 37 (2003), pp. 1–24.

[8] L. Arge, V. Samoladas, and J. S. Vitter, *On two-dimensional indexability and optimal range search indexing*, in Proceedings of the 18th ACM Symposium on Principles of Database Systems, Philadelphia, PA, 1999, pp. 346–357.

[9] L. Arge and J. Vahrenhold, *I/O-efficient dynamic planar point location*, Internat. J. Comput. Geom. Appl., to appear.

[10] L. Arge, D. E. Vengroff, and J. S. Vitter, *External-memory algorithms for processing line segments in geographic information systems*, Algorithmica, to appear.

[11] L. Arge and J. S. Vitter, *Optimal dynamic interval management in external memory*, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, Burlington, VT, 1996, pp. 560–569.

[12] R. Bayer and E. McCreight, *Organization and maintenance of large ordered indexes*, Acta Informat., 1 (1972), pp. 173–189.

[13] M. A. Bender, E. D. Demaine, and M. Farach-Colton, *Cache-oblivious B-trees*, in Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, Redondo Beach, CA, 2000, pp. 339–409.

[14] G. Blankenagel, *Intervall-Indexstructuren in Datenbanksystemen*, Informatik-Farberichte 312, Springer-Verlag, Berlin, 1992.

[15] G. Blankenagel and R. Güting, *External segment trees*, Algorithmica, 12 (1994), pp. 498–532.

[16] G. Blankenagel and R. H. Güting, *XP-Trees—External Priority Search Trees*, Technical report, FernUniversität Hagen, Informatik-Bericht 92, Hagen, Germany, 1990.

[17] N. Blum and K. Mehlhorn, *On the average number of rebalancing operations in weight-balanced trees*, Theoret. Comput. Sci., 11 (1980), pp. 303–320.

[18] Y.-J. Chiang and C. T. Silva, *I/O optimal isosurface extraction*, in Proceedings of IEEE Visualization, Phoenix, AZ, 1997, pp. 293–300.

[19] Y.-J. Chiang and C. T. Silva, *External memory techniques for isosurface extraction in scientific visualization*, in External Memory Algorithms and Visualization, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 50, J. Abello and J. S. Vitter, eds., AMS, Providence, RI, 1999, pp. 247–277.

[20] Y.-J. Chiang, C. T. Silva, and W. J. Schroeder, *Interactive out-of-core isosurface extraction*, in Proceedings of IEEE Visualization, Research Triangle Park, NC, 1998, pp. 167–174.

[21] D. Comer, *The ubiquitous B-tree*, ACM Computing Surveys, 11 (1979), pp. 121–137.

[22] H. Edelsbrunner, *A new approach to rectangle intersections, part* I, Internat. J. Comput. Math., 13 (1983), pp. 209–219.

[23] H. Edelsbrunner, *A new approach to rectangle intersections, part* II, Internat. J. Comput. Math., 13 (1983), pp. 221–229.

[24] V. Gaede and O. Günther, *Multidimensional access methods*, ACM Computing Surveys, 30 (1998), pp. 170–231.

[25] R. Grossi and G. F. Italiano, *Efficient cross-tree for external memory*, in External Memory Algorithms and Visualization, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 50, J. Abello and J. S. Vitter, eds., AMS, Providence, RI, 1999, pp. 87–106. Revised version available at ftp://ftp.di.unipi.it/pub/techreports/TR-00-16.ps.Z.

[26] R. Grossi and G. F. Italiano, *Efficient splitting and merging algorithms for order decomposable problems*, Inform. and Comput., 154 (1999), pp. 1–33.
[27] S. Huddleston and K. Mehlhorn, *A new data structure for representing sorted lists*, Acta Inform., 17 (1982), pp. 157–184.
[28] C. Icking, R. Klein, and T. Ottmann, *Priority search trees in secondary memory*, in Proceedings of Graph-Theoretic Concepts in Computer Science, Staffelstein, Germany, 1987, Lecture Notes in Comput. Sci. 314, Springer-Verlag, Berlin, 1988, pp. 84–93.
[29] P. C. Kanellakis, G. Kuper, and P. Revesz, *Constraint query languages*, in Proceedings of the 9th ACM Symposium on Principles of Database Systems, Nashville, TN, 1990, pp. 299–313.
[30] P. C. Kanellakis, S. Ramaswamy, D. E. Vengroff, and J. S. Vitter, *Indexing for data models with constraints and classes*, J. Comput. System Sci., 52 (1996), pp. 589–612.
[31] D. E. Knuth, *Sorting and Searching, Volume* 3 *of The Art of Computer Programming*, 2nd ed., Addison-Wesley, Reading, MA, 1998.
[32] E. M. McCreight, *Priority search trees*, SIAM J. Comput., 14 (1985), pp. 257–276.
[33] K. Mehlhorn, *Data Structures and Algorithms* 3*: Multi-dimensional Searching and Computational Geometry*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin, 1984.
[34] J. Nievergelt and E. M. Reingold, *Binary search trees of bounded balance*, SIAM J. Comput., 2 (1973), pp. 33–43.
[35] J. Nievergelt and P. Widmayer, *Spatial data structures: Concepts and design choices*, in Algorithmic Foundations of GIS, Lecture Notes in Comput. Sci. 1340, M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer, eds., Springer-Verlag, Berlin, 1997, pp. 153–197.
[36] M. H. Overmars, *The Design of Dynamic Data Structures*, Lecture Notes in Comput. Sci. 156, Springer-Verlag, Berlin, 1983.
[37] S. Ramaswamy, *Efficient indexing for constraint and temporal databases*, in Proceedings of the 6th International Conference on Database Theory, Delphi, Greece, 1997, Lecture Notes in Comput. Sci. 1186, Springer-Verlag, Berlin, 1997, pp. 419–431.
[38] S. Ramaswamy and P. Kanellakis, *OODB Indexing by Class Division*, A.P.I.C. Series, Academic Press, New York, 1995.
[39] S. Ramaswamy and S. Subramanian, *Path caching: A technique for optimal external searching*, in Proceedings of the 13th ACM Symposium on Principles of Database Systems, Minneapolis, MN, 1994, pp. 25–35.
[40] S. Subramanian and S. Ramaswamy, *The P-range tree: A new data structure for range searching in secondary memory*, in Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 1995, pp. 378–387.
[41] J. S. Vitter, *External memory algorithms and data structures: Dealing with MASSIVE data*, ACM Computing Surveys, 33 (2001), pp. 209–271.
[42] D. Willard and G. Lueker, *Adding range restriction capability to dynamic data structures*, J. Assoc. Comput. Mach., 32 (1985), pp. 597–617.
[43] D. E. Willard, *Reduced memory space for multi-dimensional search trees*, in Proceedings of the 2nd Symposium on Theoretical Aspects of Computer Science, Saarbrüken, Germany, Lecture Notes in Comput. Sci. 182, Springer-Verlag, Berlin, 1985, pp. 363–374.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1509–1541

COVERING RECTILINEAR POLYGONS WITH AXIS-PARALLEL RECTANGLES*

V. S. ANIL KUMAR† AND H. RAMESH‡

Abstract. We give an $O(\sqrt{\log n})$ factor approximation algorithm for covering a rectilinear polygon with holes using axis-parallel rectangles. This is the first polynomial time approximation algorithm for this problem with an $o(\log n)$ approximation factor.

Key words. approximation algorithms, covering polygons

AMS subject classifications. 52C15, 68W25, 68W40

DOI. 10.1137/S0097539799358835

1. Introduction.

We consider the problem of covering rectilinear polygons with axis-parallel rectangles. Given a rectilinear polygon P with complexity n (complexity refers to the minimum of the number of vertical edges and the number of horizontal edges in the polygon), this problem requires determining the minimum number of axis-parallel rectangles whose union covers P. The polygon P may have holes in it.

Applications. Cheng, Iyengar, and Kashyap [5] showed that this problem has applications to image compression. They claim that representing an image using a rectangle covering of its white pixels gives compression superior to that achieved by quadtrees. It also has applications to printing integrated circuits [9].

Hardness. Much effort has gone into determining the computational complexity of this problem. In spite of this, the exact complexity of this problem has remained open for many years and continues to do so. Masek [19] showed that this problem is NP-complete. Later, Culberson and Reckhow [6] used a clever reduction from 3-SAT to show that this is the case even when P has no holes. The next natural question is whether the number of rectangles needed to cover P can be computed *approximately*. Berman and Dasgupta [2] showed that this problem is MaxSNP-Hard for polygons with holes, ruling out the possibility of a polynomial time approximation scheme.

Approximation results. Note that the rectangle covering problem is a special case of the general set covering problem. Therefore, it admits an approximation algorithm with a performance guarantee of $O(\log n)$, using the greedy scheme due to Johnson [10] and Lovasz [16]. This was the best approximation factor known for the rectangle covering problem until now. Further, it is known that the general set covering problem cannot be approximated any better, modulo constant terms, unless $NP = P$ [17, 18]. However, this proof of hardness assumes certain properties about the set system which do not hold for the rectangle covering problem. In this paper, we address the issue of whether the $\Omega(\log n)$ approximation factor barrier can be broken in polynomial time for the rectangle covering problem.

There seem to be only a few examples of nontrivial algorithms breaking this

*Received by the editors July 9, 1999; accepted for publication (in revised form) March 31, 2003; published electronically October 2, 2003. An abstract of this work appeared in *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, Atlanta, GA, 1999.
http://www.siam.org/journals/sicomp/32-6/35883.html
†CCS-5, Los Alamos National Laboratory, Los Alamos, NM 87544 (anil@lanl.gov). This work was done while this author was a graduate student at the Indian Institute of Science.
‡Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India 560012 (ramesh@csa.iisc.ernet.in).

barrier for specific instances of the set covering problem. Brönnimann and Goodrich [3] showed that the covering problem for any set system with Vapnik–Chervonenkis dimension d can be approximated within an $O(d \log(dc))$ factor, where c is the cost of the optimal covering. For the rectangle covering problem, while d is a constant, c can be shown to be $\Omega(\sqrt{n})$; therefore, the Brönnimann–Goodrich algorithm gives only an $O(\log n)$ approximation factor. Brönnimann and Goodrich [3] also showed that $O(c)$ sized set covers can be computed for two-dimensional disc covering and a problem related to three-dimensional polytope separation as these set systems admit ϵ-nets of small size. It is not clear whether the rectangle covering problem admits ϵ-nets of small size.

Special situations. There are special situations when the above barrier can indeed be broken. When P is hole-free, Franzblau [7] showed a factor 2 approximation guarantee. When P has holes, Franzblau also gave an $O(n \log n)$ time heuristic which gives an $O(\log n)$ approximation factor. When P is both vertically and horizontally convex (i.e., the intersection of any vertical or horizontal line and P is just a single line segment), Chaiken et al. [4] gave a polynomial time algorithm which computed the minimum number of rectangles required, exactly. This was improved upon by Franzblau and Kleitman [8], who achieved the same result under the weaker restriction that P is just vertically convex. Note that both restrictions preclude the presence of holes.

Other papers which have dealt with this problem are [9, 11, 12, 13, 5].

Our result. We give the first algorithm to break the $\Omega(\log n)$ barrier even when P has holes. Our algorithm gives an approximation guarantee of $O(\sqrt{\log n})$.

Our algorithm is in fact trivial, and our contribution lies entirely in showing a lower bound. For simplicity, assume that all holes in P are point holes.[1] Then our algorithm simply puts one rectangle for each *strip*, i.e., a stretch of points between two vertically aligned holes (e.g., strip s and its associated rectangle R in Figure 1; this is defined formally in section 2). This rectangle covers the strip entirely and is made as thick as possible. It is easy to see that the rectangles for all strips together cover P (the boundary of P must be treated as being lined by point holes for this). Also note that the rectangles for two distinct strips could be identical (e.g., strips B and D in Figure 2). We show that the total number of distinct rectangles $\#N$ obtained in the above process is $O(\sqrt{\log n} * |OPT|))$, where OPT is the minimum cover.

The lower bound. The main hurdle in breaking the $\Omega(\log n)$ barrier is to obtain a good lower bound for the optimum. One such lower bound is the cardinality of the largest *independent set* or *antirectangle*, i.e., a set of points in P, no two of which can be covered by the same rectangle. Chvatal (as reported in [4]) originally conjectured that the size of the minimum covering equals the size of the largest independent set. While this is indeed true for vertically and horizontally convex P, as shown by Chaiken et al. [4], it is not true for general P, with or without holes. Szemeredi found a counterexample with holes, and Chung found one without holes (both reported in [4]). Erdos (as reported in [4]) asked whether the ratio of the sizes of the minimum covering and the largest independent sets is bounded. It is easy to show that this ratio is $O(\log n)$. However, to the best of our knowledge, the best lower bound on this ratio known to date is just $21/17 - \epsilon$, due to [4].

Instead of using the above independent set bound, we use the *clique covering* lower bound. Consider the finite set of all points in P after suitable discretization.

[1] The case when the holes are arbitrary can effectively be reduced to the case of point holes, as we will show later in this paper.

Consider the graph G with these points as vertices and an edge between two points if and only if they are both covered by some rectangle. It is easy to see that $|OPT|$ is exactly the size of the smallest clique cover of this graph (i.e., a collection of cliques that covers all vertices). We shall lower bound the clique cover number by obtaining an upper bound on the sizes of cliques in OPT.

One problem we face in the process is that the cliques of G could be very different in size and therefore do not admit a uniform upper bound. However, we show that if none of the strips are *jumpers* (we will define this term in section 3.2), then we can choose $\Theta(\#N)$ points such that the maximum clique size in the subgraph induced by them is $O(\sqrt{\log n})$, and therefore $|OPT| = \Omega(\#N/\sqrt{\log n})$, as required. Our proof of this fact is based on a somewhat detailed exploration of the structure of cliques in G.

There is a correspondence between the points that are chosen in the above induced subgraph and the set of strips. We choose two points for each strip, one to its left and one to its right, after partitioning the strips into disjoint sets called *families* (defined later); this ensures that the induced subgraph has $\Theta(\#N)$ points. As mentioned earlier, if there are no jumpers, we show that the maximum clique size is $O(\sqrt{\log n})$ (this is not strictly true; we show that a further subset of this set of points has this property). Also, the number of jumpers turns out to be at most $(|OPT|\sqrt{\log n})$, so we can ignore such strips and argue about the points defined by the rest. The above description is very incomplete and will be developed formally in the remaining sections.

A significant point to note is that while we show a lower bound on the clique covering number of graph G in the absence of jumpers, we are unable to show a good lower bound on the size of the largest independent set. We can show that the number of families is a lower bound on the size of the largest independent set; however, the average family size can be as large as $\Omega(\log n)$, as we shall show later.

Roadmap. In section 2, we show how to discretize the polygon and then describe our algorithm for laying rectangles. For simplicity and clarity, we first explain the arguments in section 3 under some restrictive assumptions. The general case requires a refinement of these ideas and is handled in section 4. Section 5 describes an example where the average family size is large. Section 6 mentions the loose threads which remain in this problem and also describes some related problems.

2. Preliminaries. Consider the grid formed by drawing infinitely long lines through each vertical and horizontal edge of the polygon (i.e., both the polygon boundary and the hole boundaries) (see Figure 1). Note that this need not be a uniform grid; the spacing between adjacent grid lines is not necessarily the same. Let n denote the *vertical complexity* of P, i.e., the number of horizontal grid lines. Without loss of generality, we assume that the vertical complexity is at most as large as the horizontal complexity.

Viewing the entire plane as partitioned into grid cells, the term *hole* shall henceforth denote any grid cell which is in the exterior of the polygon (i.e., either outside the outer boundary or within one of the holes). Since any grid cell lies either completely in the interior of the polygon or completely in the exterior of the polygon (see Figure 1), the above term is well defined.

Note that any two holes are either perfectly aligned or completely misaligned with respect to the grid lines. Also note that two holes could lie side by side, touching each other (e.g., holes A, B in Figure 1). If we treat each grid cell as a point, we get the case of point holes. However, the rest of the description will be in terms of cells.

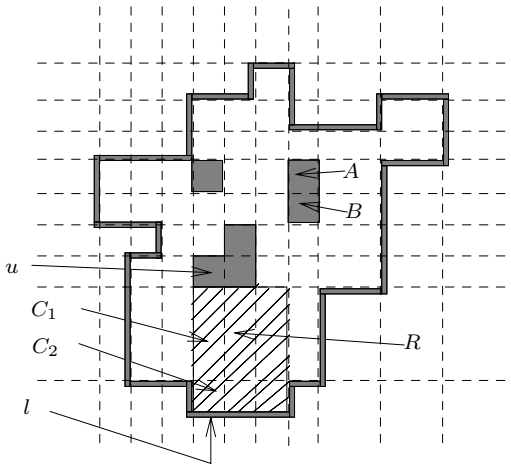


FIG. 1. *The grid: Dark cells and external cells are holes.* C_1, C_2 *form a strip* s *of length* 2. R *is the rectangle associated with* s*, and* u *and* l *are its upper and lower holes, respectively.*

2.1. Defining strips and laying rectangles. A sequence of consecutive vertically aligned nonhole cells bounded by holes on the top and the bottom constitutes a *strip* (see Figure 1). The *length* of a strip A is the number of cells in it and is denoted by $l(A)$. The *upper hole* for a strip is the hole which lies immediately above the topmost cell for that strip. The *lower hole* is similarly defined.

For each strip, we define its *associated rectangle* to be the unique rectangle that covers this strip and extends as far as possible to the left and to the right. In other words, the associated rectangle is obtained by sweeping the strip to the left and right until it is blocked by some holes on both sides (see Figure 1). The algorithm simply adds the associated rectangle of every strip to the overall cover. Lemma 2.1 shows that these rectangles indeed cover the given polygon. The rest of the paper proves that this naive way of covering is at most an $O(\sqrt{\log n})$ factor larger than the optimum.

The hole which blocks the associated rectangle of a strip S on the right is called the *right blocking hole* of S, with ties broken in favor of the topmost hole. *Left blocking holes* are defined similarly.

LEMMA 2.1. *Each point in the polygon is contained in the associated rectangle of some strip.*

Proof. Each nonhole grid cell, c, belongs to a unique strip. This is because if the cell c is swept vertically up and down, it would hit a hole in both directions. The rectangle associated with this strip contains c. □

Note that two strips could have identical associated rectangles (e.g., strips B, D in Figure 2). Consider equivalence classes of strips, where strips with the same associated rectangle are in one class. All but the rightmost of the strips in an equivalence class are called *unnecessary strips*. Clearly, an unnecessary strip can be ignored. It suffices to account for the rectangles associated with necessary strips.

2.2. Spanning, nestedness, and disjointness. A rectangle associated with strip A is said to *pass through* strip B if some but not all of the cells in strip B are contained in this rectangle (see Figure 2). In this situation, B is said to *l-span* A if it is to the left of A and *r-span* A if it is to the right of A. Note the strict condition in the above definition; i.e., the length of B must be strictly greater than that of A.

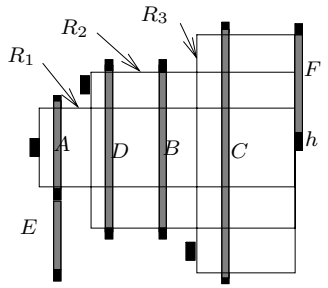


FIG. 2. *A family: Strips A, B, C are in one right family and have the common blocking hole h. D is unnecessary and has the same associated rectangle as B. B r-spans A. Strips E and F are disjoint.*

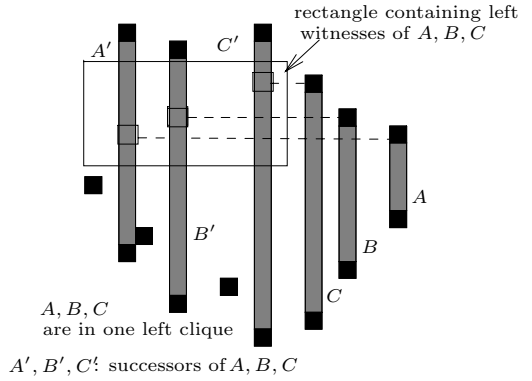


FIG. 3. *A left clique. B l-spans A, and C l-spans B.*

A collection of strips is called *right nested* if the strips increase in length from left to right and each strip r-spans all smaller strips. Left nesting is defined analogously. The strips A, B, and C in Figure 3 are left nested.

Two strips A, B are said to be *disjoint* if one of the following conditions holds: (i) the lower hole of A is horizontally aligned with or higher than the upper hole of B, or (ii) the upper hole of A is horizontally aligned with or below the lower hole of B (see Figure 2).

2.3. Successors, terminals, and witness cells. We bound the approximation factor of our algorithm by first identifying a subset of the nonhole cells and then showing lower bounds on the number of rectangles needed to cover these cells. The nonhole cells we identify are called *witness* cells. There are two kinds of witness cells, *left witness* cells and *right witness* cells. To define witness cells, we need the notion of successor strips.

We will formally define successor strips in section 3.1. Here, we introduce some properties of successors. The *right successor* strip for a strip A r-spans A, and the *left successor* for a strip A l-spans A. Not each necessary strip has a right successor; those strips which do not have right successors are called *right terminal* strips. *Left terminal* strips are defined analogously.

Given successors and terminals, witnesses are defined as follows. For each right terminal strip A, its right witness cell is the cell in A which is horizontally aligned with its right blocking hole (see Figure 4). For each right nonterminal strip A, its

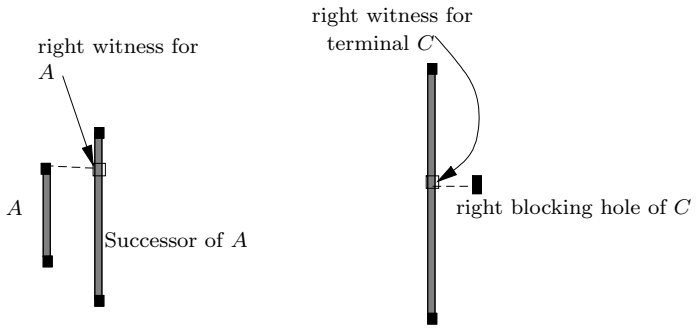


FIG. 4. *Right witnesses for nonterminal strip A and terminal strip C.*

right witness is the cell in its right successor strip which is horizontally aligned with A's upper hole (see Figure 4). The left witness is defined in a similar manner.

Note that right witnesses for right nonterminal strips A are not well defined if the upper hole of the right successor of A is aligned with the upper hole of A. An analogous fact holds for left witnesses of left nonterminal strips. This is clearly a problem because the above process may not define sufficiently many witness cells to obtain a good enough lower bound on the size of the optimum. We solve this problem by first identifying a constant fraction of the necessary strips with the following property: each such strip has a well-defined left witness. The remaining necessary strips are called *discarded* strips, and they play no role in the proof.

The precise definition of successor strips and the description of which strips are discarded appear in subsequent sections. We set up some more preliminaries in this section.

2.4. Cliques and the optimum cover. The optimum cover, denoted by OPT, must cover all the witness cells defined above. Consider a graph G whose vertices are the various witness cells defined above and whose edges denote that the two associated cells can be covered together by a single rectangle. Note that each cell is either completely inside or completely outside any maximal rectangle (one which has holes touching all four sides). Two witness cells are said to be *independent* if no single rectangle covers both of them; i.e., there is no edge between them in G. The following lemma holds.

LEMMA 2.2. *All witness cells contained in any rectangle form a clique in G. Conversely, any clique in G comprises witness cells which can be covered by just one rectangle.*

Proof. The first statement of the lemma follows from the definition of G.

We now show that any clique C in G can be covered by one rectangle. Let a, b, c, d be the leftmost, topmost, rightmost, and bottommost witness cells in C, respectively. Note that a, b, c, d need not be all distinct. We claim that the rectangle R having a, b, c, d on its left, top, right, and bottom edges, respectively, is hole-free (see Figure 5). The claim then follows since a, b, c, d are extreme points.

To show that R is hole-free consider all edges between witness cells a, b, c, d. If a, b, c, d are all distinct there will be six such edges, and fewer otherwise. There exist up to six rectangles, each of which covers both witness cells for one of the above edges (see Figure 5). Clearly, R is contained in the union of these rectangles. □

We partition the witness cells into $O(|OPT|)$ cliques, where $|OPT|$ denotes the number of rectangles used by OPT, in the following manner. For each witness cell,

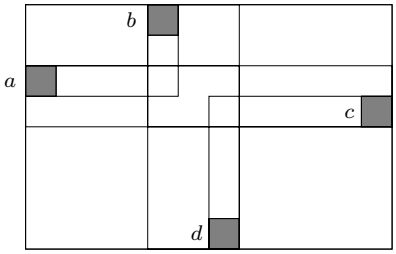


FIG. 5. *The rectangle R formed by a, b, c, d and the six possible rectangles formed by each pair of witness cells, which together cover R.*

assign it to some rectangle covering it in OPT, breaking ties arbitrarily. From Lemma 2.2 above, this corresponds to a partition of witness cells into cliques. We partition each clique further into two parts, one containing left witness cells and another containing right witness cells; these parts are called the *left clique* and the *right clique*, respectively.

Remark. We say that a strip is *in* a particular left clique (right clique, respectively) if its left witness cell (right witness cell, respectively) is in that clique, and, by abuse of notation, we will sometimes identify a strip with its witness cell.

To bound the approximation factor of our algorithm, we will show a lower bound on the number of cliques obtained above. This lower bound will exploit several interesting properties of these cliques. However, before we describe these properties, we need to specify how successors are determined and how strips to be discarded are identified. We start by describing the above for a simpler special case so as to bring out the intuition behind our proof.

We again remind the reader that unnecessary strips are being ignored, and any reference to a strip in the rest of the paper denotes a necessary strip.

3. The lower bound argument: A special case. We make the following assumptions in this section and illustrate the main ideas of the proof of the lower bound for this special case. We shall return to the general case in section 4.

Assumption 1. The length of each strip is a power of 2.

Assumption 2. The upper hole of a strip is not horizontally aligned with that of its left or right successor strip. The notion of a successor strip was introduced in section 2, and successors will be defined shortly.

Recall the preceding discussion on discarded strips in section 2. Assumption 2 precludes exactly those situations which forced us to introduce the notion of discarded strips. It follows that there is no need to discard any strips. Thus, all necessary strips will have associated left and right witnesses. Assumption 1 will make the definition of successor strips a little easier.

3.1. Defining families and successors. We organize strips into families as follows. Our proofs crucially exploit the interplay between cliques and families.

We define a *right family* to be a set of strips with the same right blocking hole (Figure 2). It is easy to see that strips in a right family are right nested (the strict increase in strip lengths from left to right is a consequence of the absence of unnecessary strips). The *right successor* of a strip A in a right family is defined as the next strip A' to the right in the family. The rightmost strip in the family does not have a successor and is a *right terminal strip*. Left families, successors, and terminal strips

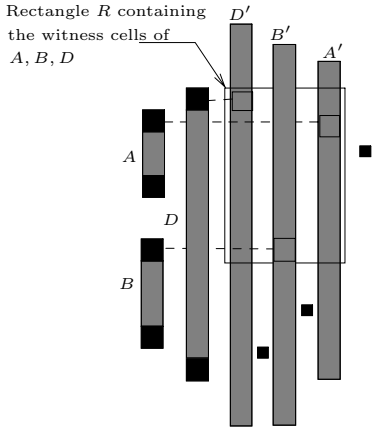


FIG. 6. *A, B, D are in a right clique. A', B', D' are their respective right successors. A is an exception.*

are defined analogously. It follows from Assumption 1 and the nestedness property that the number of strips in a family is at most $\log n$.

Remark. The notion of successor defined above will have to be changed when we deal with the general case in section 4 in the following manner. We will define two types of strips and use the above definition for strips of the first kind; strips of the second kind will require a different definition, and the successor of such a strip A will have length $< 2l(A)$. However, it will continue to be the case that the right successor of A r-spans A, and, similarly, the left successor of A l-spans A for all undiscarded strips A. Note that Assumption 1 precludes the existence of strips of the second kind above.

3.2. Properties of cliques. Our proof is based on some structural facts, which we state in the following lemmas. In section 3.3, we will use these lemmas to obtain the approximation factor. The proofs of these lemmas appear in section 3.4.

LEMMA 3.1. *All right (left, respectively) witness cells associated with right (left, respectively) terminal strips are independent. Therefore, the number of families and terminal strips is $O(|OPT|)$.*

All further references to strips in subsequent lemmas in this section will be to nonterminal strips.

LEMMA 3.2. *Strips in a right (left, respectively) clique belong to distinct right (left, respectively) families.*

LEMMA 3.3. *With the exception of at most one strip, all strips in a right (left, respectively) clique constitute a right (left, respectively) nested set of strips.*

Thus, each clique of OPT has at most two exceptions, one in each direction. The total number of exception strips is therefore $O(|OPT|)$. These exception strips can be removed from consideration. Figure 6 shows an example of an exception strip in a right clique. All further references to cliques in this section will assume that exception strips are not present.

LEMMA 3.4. *Strips in any right (left, respectively) clique are in distinct length categories; i.e., if a particular strip has length in the range $[2^i, 2^{i+1})$, then the next strip to the right (left, respectively) has length at least [2] 2^{i+1}.*

[2]Strips in a clique will actually at least double in length by Assumption 1. However, we prefer

In addition, for any $x > 0$, the number of strips in a right (left, respectively) clique or a right (left, respectively) family whose length is at least 2^x times the length of one of the two previous strips to the left (right, respectively) is $O(\frac{\log n}{x})$.

LEMMA 3.5. *Let A, B be strips in a particular right (left, respectively) clique, with $l(A) < l(B)$. Let A', B' be the right (left, respectively) successors of A, B, respectively. These four strips must be in the following order from left to right (right to left, respectively): A, B, B', A'. In addition, A' cannot r-span (l-span, respectively) B and must have its upper hole above that of B.*

DEFINITIONS. A strip A is called a *right jumper* if its right successor has length at least $2^{\Delta} l(A)$, where Δ is a parameter. This parameter will be set to $\Theta(\sqrt{\log n})$ at the end. Left jumpers are defined analogously. The *vertical separation* between two holes a, b is the vertical distance between their lower boundaries, measured in terms of the number of grid cells (see Figure 12).

LEMMA 3.6. *Let A, B be nonjumper strips in some right (left, respectively) clique, with $l(A) < l(B)$. The following two facts hold.*

1. *The vertical separation between the upper holes of A and B is at most $2^{\Delta} l(A)$.*
2. *If A is not amongst the smallest $\gamma\Delta$ nonjumper strips in this right clique (left clique, respectively), the vertical separation between the upper holes of A and B is at most $\frac{l(B)}{2^{(\gamma-1)\Delta}}$.*

LEMMA 3.7. *Let A and B be strips belonging to the same right (left, respectively) clique, with $l(A) < l(B)$. Then A lies completely above the right (left, respectively) blocking hole of B.*

LEMMA 3.8. *Let C and C' be the left and right cliques, respectively, containing the left and right witness cells, respectively, of strip A. Let B be a strip in C smaller than A. Let B' be a strip in C' smaller than A. Then B' cannot l-span B, and B cannot r-span B'.*

3.3. Accounting for strips. The number of terminal strips is $O(|OPT|)$ by Lemma 3.1. The number of jumper strips is $O(|OPT| * \frac{\log n}{\Delta})$ by Lemma 3.4. All references to strips in the rest of this section are to nonterminal, nonjumper, nonexception (see Lemma 3.3) strips. All references to cliques assume that terminal, jumper, and exception strips have been removed; references to clique sizes denote sizes subsequent to this removal.

We now consider the remaining strips and show that there exists a large subset W of the witnesses associated with these strips such that the graph induced by W has only small, i.e., size $\Theta(\Delta)$, cliques. A rough reason why such a subset W exists is as follows.

By the nestedness property of strips in a clique and by Assumption 1, large cliques will necessarily have long strips and will therefore require proportionately large vertical space. In addition, Lemma 3.6 states that if A, B are strips in a large clique, then the vertical separation between the upper holes of A and B is small. Given the properties of cliques stated above, we will show that the left cliques containing A, B and the right cliques containing A, B cannot all satisfy the dual requirements of large vertical space and small vertical separation, unless one of these cliques is small. This intuition is formalized below.

Remark. We mention here that the rest of this section uses only Lemmas 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, and 3.8. Assumptions 1 and 2 will not be used directly. When we drop these assumptions and proceed to the general case, we will apply the

---

to work with this weaker condition, as it generalizes even when Assumption 1 is dropped.

description below to a carefully chosen subset of strips for which the above lemmas will indeed hold.

DEFINITION. The *right follower* of a strip B is defined to be the unique next longer strip, if any, in the right clique containing B (uniqueness holds from Lemma 3.3 and since exceptions are ignored). In Figure 6, D is a follower of B. Strip A also has D as its follower, but A is dropped as it is an exception strip. *Left followers* are defined similarly.

LEMMA 3.9. *Let P, Q be strips in some right clique C, with Q being the right follower of P (Q need not exist). Then one of the following must hold.*

1. *P is among the smallest 3Δ strips in C, or P is the largest strip in C.*
2. *Q is among the smallest $5\Delta+1$ strips in C'', where C'' denotes the left clique containing Q.*
3. *$2^{\Delta} l(P) \leq l(Q)$.*
4. *Let C' denote the left clique containing P. Let P' be the strip in C' whose left follower is P. Either P' does not exist or $2^{\Delta} l(P') \leq l(P)$.*
5. *Let $Q_1, \ldots, Q_k$, in increasing order of length, be the strips in C'' which are smaller than Q. Then $2^{2\Delta} l(Q_{k-1}) \leq l(Q)$.*

Analogous statements hold for strips P, Q in a left clique C.

Proof. We suppose that none of the above five conditions holds to get a contradiction.

Since condition 1 is not satisfied, Q exists. Since condition 2 is not satisfied, $k \geq 5\Delta+1$. By Lemma 3.8, none of $Q_1, \ldots, Q_k$ r-span P (see Figure 7(a)). Similarly, P cannot l-span any of $Q_1, \ldots, Q_k$. By Lemma 3.3, Q r-spans P, and l-spans each of $Q_1, \ldots, Q_k$; therefore, the hatched regions must be hole-free. We consider two cases now, depending upon whether the upper hole of Q_k is above or below that of P.

First, suppose the upper hole of Q_k is aligned with or above that of P (see Figure 7(a)). The lower hole of Q_k must be above that of P; otherwise, Q_k r-spans P, which contradicts Lemma 3.8. Therefore, the left blocking hole of Q_k will be horizontally aligned with or above the upper hole of P. By Lemma 3.7, Q_{k-1} is completely above this blocking hole and therefore completely above P. Since the fifth condition is not satisfied, $2^{2\Delta} l(Q_{k-1}) > l(Q)$. Then the vertical separation between P and Q is at least $l(Q_{k-1}) > \frac{l(Q)}{2^{2\Delta}}$. Since the first condition is not satisfied, Lemma 3.6 implies that the vertical separation between P and Q is at most $\frac{l(Q)}{2^{2\Delta}}$, a contradiction. While applying Lemma 3.6, recall that jumpers have been excluded from cliques earlier.

Second, suppose the upper hole of Q_k is below that of P (see Figure 7(b)). The bottom hole of Q_k must be below that of P; otherwise, P will l-span Q_k, which contradicts Lemma 3.8. Since Q_k l-spans each of $Q_1, \ldots, Q_{k-1}$, these must also have their upper holes below that of P. Since P cannot l-span $Q_1, \ldots, Q_k$ (by Lemma 3.8), their lower holes must also be below that of P. Since condition 4 is violated, P' exists. By Lemma 3.3, P l-spans P', and therefore P' must be to the right of Q (see Figure 7(b)). Since P l-spans P', Q l-spans each of $Q_1, \ldots, Q_k$ (by Lemma 3.3), and lower holes of $Q_1, \ldots, Q_k$ are below that of P, it must be the case that each of $Q_1, \ldots, Q_k$ either l-spans P' or is completely below it (Q_k l-spans P', while Q_i lies below P' in Figure 7(b)). We claim that at most 2Δ of $Q_1, \ldots, Q_k$ can l-span P'. This is shown in the next paragraph. Then the vertical separation between the upper holes of $Q_{k-2\Delta}$ (which is completely below P') and Q is at least $l(P') > \frac{l(P)}{2^{\Delta}} > \frac{l(Q)}{2^{2\Delta}}$ (because conditions 3 and 4 are not satisfied). Since $k - 2\Delta > 3\Delta$, Lemma 3.6 applied to $Q_{k-2\Delta}$ and Q implies that the vertical separation between the upper holes of these two strips is at most $\frac{l(Q)}{2^{2\Delta}}$, a contradiction.

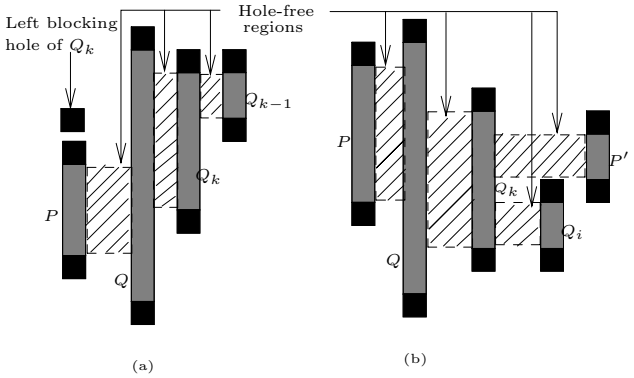


FIG. 7. *Two situations for the upper hole of* Q_k.

It remains to show that at most 2Δ of $Q_1, \ldots, Q_k$ can l-span P'. We show that $Q_{k-2\Delta}$ cannot l-span P'. Suppose this is not true. Then $Q_{k-2\Delta}, \ldots, Q_k$ all l-span P' and $l(Q_{k-2\Delta}) > l(P')$. By Lemma 3.4, $Q_{k-2\Delta}, \ldots, Q_k$ are in distinct length categories, and therefore $l(Q) \geq 2^{2\Delta} l(Q_{k-2\Delta}) > 2^{2\Delta} l(P') > 2^{\Delta} l(P)$. The last inequality follows from the violation of condition 4. Then condition 3 is satisfied, a contradiction. □

COROLLARY 3.10. *The number of nonterminal, nonjumper, nonexception strips is* $O(|OPT| * (\frac{\log n}{\Delta} + \Delta))$.

Proof. Each nonterminal, nonjumper, nonexception strip P must be in some right clique C and in some left clique C'.

We consider five classes of these strips, depending upon which of the conditions in Lemma 3.9 is satisfied. The number of strips P which satisfy the first condition is clearly $O(|OPT| * \Delta)$ because each clique in OPT has $O(\Delta)$ such strips. The number of strips P which satisfy the third condition is $O(|OPT| * (\frac{\log n}{\Delta}))$ by Lemma 3.4. Similarly, the number of strips P which satisfy the fourth condition is $O(|OPT| * \frac{\log n}{\Delta})$. Next, consider strips P which satisfy either condition 2 or 5. Such a strip P has a unique right follower Q. Note that any strip is the right follower of at most one strip. Thus it suffices to bound the number of strips Q which are right followers of strips P satisfying condition 2 or 5. Using the same argument as for condition 1, the number of strips Q satisfying condition 2 is $O(|OPT| * \Delta)$. Using an argument similar to that for condition 4, the number of strips Q satisfying condition 5 is $O(|OPT| * \frac{\log n}{\Delta})$. □

Thus, given Assumptions 1 and 2, by setting $\Delta = \Theta(\sqrt{\log n})$ we get that the number of rectangles laid out by our algorithm is within an $O(\sqrt{\log n})$ factor of the optimal.

3.4. Proofs. We give the proofs of Lemmas 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, and 3.8 in that order. For the general case, we will not repeat the proofs of these lemmas. To convince the reader that these proofs would continue to hold even in the absence of Assumptions 1 and 2, we describe the proofs so that they are dependent only on the following facts and on proofs of previous lemmas in the above order, instead of Assumptions 1 and 2 directly. These facts are consequences of Assumptions 1 and 2 and of the way in which families and successors were defined. Thus, as long as the general case obeys these facts, and if we derive the generalizations of these lemmas in the same order, these proofs will continue to hold. However, there is one exception, namely the first part of Lemma 3.4, where we shall use Assumption 1. This shall

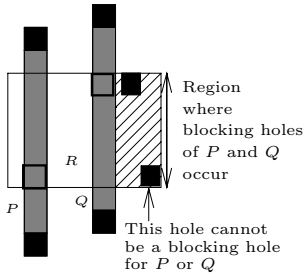


FIG. 8. *Witnesses of terminal strips are independent.*

require reproving when we get to the general case. This will be made precise in section 4.4.

We state Facts 1–6 here.

FACT 1. *Strips in a right (left, respectively) family are right (left, respectively) nested and have the same right (left, respectively) blocking hole. Further, strips in distinct right (left, respectively) families have distinct right (left, respectively) blocking holes.*

FACT 2. *Each family has one terminal strip. Further, strips in a family lie in distinct length categories (length categories are given by the length ranges* $[2^i, 2^{i+1})$, $1 \leq i \leq \log n - 1$*).*

Note that by Assumption 1*, strips in a family satisfy a stronger property; namely, they at least double in size. However, the weaker property stated above will be all that is available in the general case.*

FACT 3. *The right (left, respectively) successor of a strip A is the next strip to the right (left, respectively) in the right (left, respectively) family containing A.*

FACT 4. *The right (left, respectively) successor of a strip A r-spans (l-spans, respectively) A.*

FACT 5. *The right (left, respectively) witness of a nonterminal strip A, if it exists, is horizontally aligned with the upper hole of A and lies on the right (left, respectively) successor of A.*

Note that the condition "if it exists" always holds by Assumption 2*. However, this will not be true in the general case, after Assumptions* 1, 2 *are dropped.*

FACT 6. *The right (left, respectively) witness of a terminal strip lies on the terminal strip itself and is horizontally aligned with its right (left, respectively) blocking hole.*

We now give the proofs. At the end of each proof below, we make a careful note of which of the above facts are used.

Proof of Lemma 3.1. We give the proof for right terminals. The proof for left terminals is similar.

Suppose for a contradiction that there are two right terminal strips P, Q whose right witnesses are not independent. P and Q must be in distinct right families, and their right blocking holes are distinct, by Facts 1 and 2. Recall that the right witness cells of P and Q are in P and Q, respectively, and are horizontally aligned with their respective right blocking holes (see Fact 6).

Consider a rectangle containing the two witness cells; such a rectangle exists by Lemma 2.2 (see Figure 8). Clearly, both P and Q must have their upper holes above and lower holes below this rectangle. Without loss of generality, assume P is to the left of Q. Note that the right blocking holes for both P and Q must be in the hatched

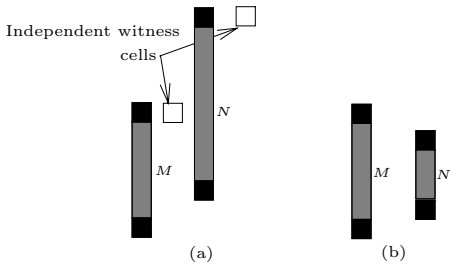


FIG. 9. *Two situations for M, N in Lemma 3.3 if N neither r-spans M nor is disjoint from M.*

region (i.e., the extension of the above rectangle to the right; the rectangle itself must be hole-free). Therefore, both blocking holes must be to the right of Q. Then it follows that whichever one of these holes occurs further to the right of the other cannot be the right blocking hole for either P or Q. In addition, if the two blocking holes are vertically aligned, the lower one cannot be the right blocking hole for either P or Q.[3] This gives a contradiction. □

Remark. Note that the above proof used only Facts 1, 2, and 6 among the six facts listed above.

Proof of Lemma 3.2. We show that if A and B are two strips in the same right family, their right witness cells are independent. Similar arguments hold for the right.

Without loss of generality, assume that $l(A) < l(B)$. By Fact 1, B is to the right of A and r-spans A. By Facts 3 and 5, A defines its right witness cell either on B (if B is the right successor of A) or to the left of B. Since B defines its right witness cell on the horizontal line containing the upper hole of B and to the right of the upper hole of B, it is easy to see that any rectangle containing the right witness cells of A and B has to contain the upper hole of B. This implies that these two witness cells are independent and cannot be in a clique. □

Remark. Note that the above proof used only Facts 1, 3, and 5 among the six facts listed above.

Proof of Lemma 3.3. We prove the lemma for a right clique C; similar arguments hold for left cliques. The following fact will be useful.

FACT. Suppose strips M and N are two strips in C, and M is either to the left of N or vertically aligned with it. Then either M and N are disjoint or N r-spans M. This must be true; otherwise, one of the two situations shown in Figure 9 holds, and then M's right witness cell will be independent from the right witness cell of N (recall Facts 4 and 5).

Suppose the strips in C are not right nested. By the above fact, if each strip in C r-spans the smallest strip in C, then there are no disjoint strips in C, and the strips in C must be right nested. So there must exist a strip in C which is disjoint from the smallest strip A in C; consider the smallest such strip B. Clearly, neither A nor B can r-span any strip in C. We will show that all other strips D in C must r-span the lower of A, B. It would then follow from the above fact that the strips in C with the upper of A, B removed are right nested. We consider the case when B is below A; the other case is similar.

Consider a strip D in C, other than the strips A, B, and suppose for a contradiction that D does not r-span B. Recall from the previous paragraph that neither

[3]Here, we use the fact that ties for the blocking hole were broken in favor of the upper hole.

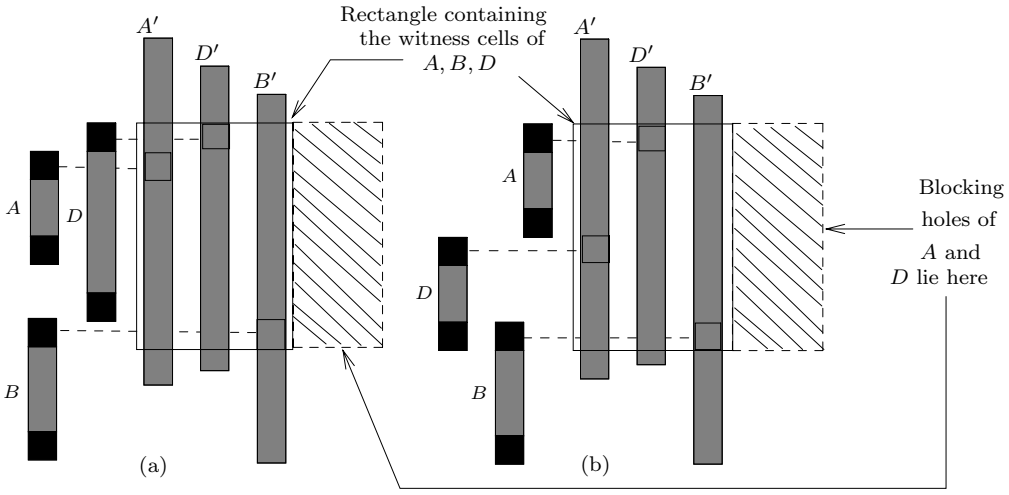


FIG. 10. (a) *D r-spans A but not B.* (b) *A, B, D are disjoint. The order in which the successors A', B', D' are present is not important.*

A nor B r-span any other strips in C. By the above fact, D must be disjoint from B. Further, D either is disjoint from A or is to the right of A and r-spans A (see Figure 10(a),(b)). We will show in the next paragraphs that the right blocking holes of the two upper strips among A, B, D must be identical. But, by Lemma 3.2, A, B, D must all be in distinct right families, and therefore, by Fact 1, they have distinct right blocking holes, a contradiction.

To show that the right blocking holes of the two upper strips among A, B, D must be identical, consider the rectangle R associated with clique C (by Lemma 2.2, such a rectangle exists). This rectangle has the following properties. The right witness cells for A, B, D are all in R, and A, B, D are themselves to the left of R. By Fact 5, the right witness cells of A, B, D are aligned with their respective upper holes and lie on their respective right successors, A', B', D'. By Fact 4, A', B', D' r-span A, B, D, respectively. Therefore, R must have its upper edge above the upper holes of A, B, D and its lower edge below all these holes. In addition, A', B', D' must all have their upper holes above R and bottom holes below R (see Figure 10); these successor strips must stab vertically through R. Note that the relative placement of A', B', D' is not important, though Figure 10 shows D' placed between A' and B'. Recall again that A, B are disjoint, D, B are disjoint, and D either r-spans A or is disjoint from A.

It follows from these properties that the right blocking holes of the upper two strips X, Y amongst strips A, B, D must be in the hatched region, i.e., the right extension of R. Next, since a strip and its right successor must have the same right blocking hole by Facts 1 and 3, the leftmost hole in the hatched region must be the right blocking hole for both X, Y, as required. □

Remark. Note that the above proof directly used only Facts 1, 3, 4, and 5 among the six facts listed above, and Lemma 3.2.

Proof of Lemma 3.4. Strips in a right (or left) clique strictly increase in length by Lemma 3.3. The first part of the lemma then follows from Assumption 1.

Next, we prove the second part of Lemma 3.4. Right families are right nested by Fact 1. By Lemma 3.3, right cliques are also right nested. Therefore, in both right

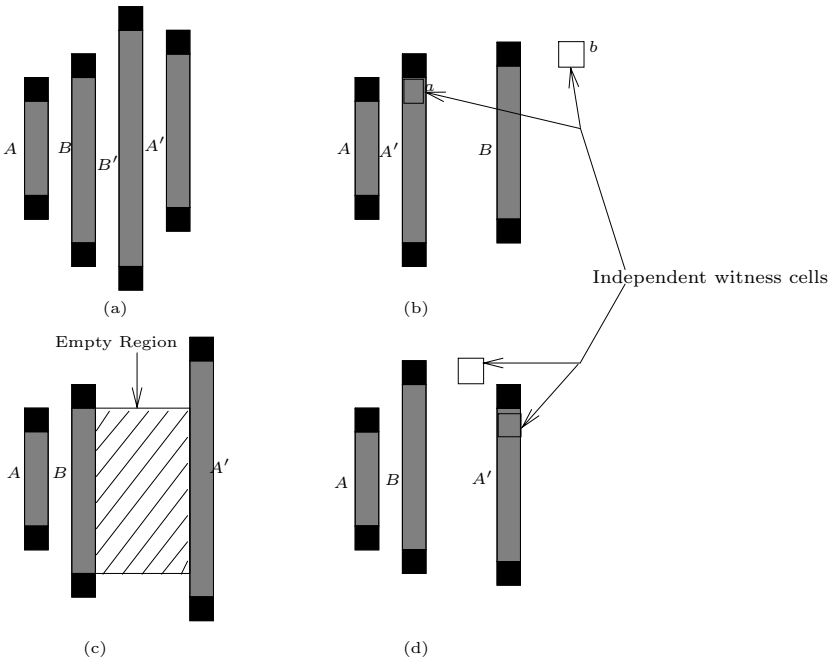


FIG. 11. (a) *B and B′ lie between A and A′.* (b) *A′ lies to the left of B.* (c) *A′ r-spans B.* (d) *Upper hole of A′ lies below that of B.*

families and right cliques, strip lengths increase strictly monotonically to the right. Since the smallest strip length is 2^0 and the largest is $2^{\log n}$, the lemma follows. A similar argument holds for left families and cliques. □

Remark. Note that the above proof directly used Assumption 1, only Fact 1 among the six facts listed above, and Lemma 3.3.

Proof of Lemma 3.5. We prove the lemma for a right clique C. The argument for left cliques is similar.

From Lemma 3.3, it follows that B r-spans A and hence must lie to the right of A. If B is to the right of A', then the right witness cells a and b of A and B, respectively, are independent (see Figure 11(b); also see Facts 4 and 5). Therefore, B is to the left of A' and to the right of A. To show that the right successor of B also lies between A and A', we show in the next paragraph that A' cannot r-span B. Since A' and B both r-span A, they cannot be disjoint either. Then it follows that the right blocking hole of B is vertically aligned with or to the left of A'. Therefore, the right successor of B is also to the left of A' (since, by Facts 1 and 3, a strip and its right successor have the same right blocking hole).

Suppose A' r-spans B. Then the right blocking hole of B would be the same as that of A (which is identical to that of A' by Facts 1 and 3) (see Figure 11(c); the hatched region must be hole-free); then A and B would be in the same right family by Fact 1. This contradicts Lemma 3.2.

It remains to show that the upper hole of A' is above that of B. By Fact 5, B's right witness cell lies on the horizontal line containing the upper hole of B. If the upper hole of A' is horizontally aligned with or below that of B, the witness cells a and b of A and B, respectively, would be independent (see Figure 11(d)). Hence, the upper hole of A' lies above that of B. □

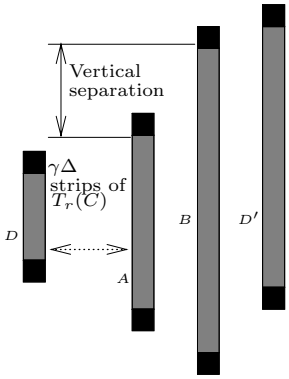


FIG. 12. *Vertical separation in a right clique.*

Remark. Note that the above proof directly used only Facts 1, 3, 4, and 5 among the six facts listed above, and Lemmas 3.2 and 3.3.

Proof of Lemma 3.6. We prove the lemma for a right clique C. An analogous argument holds for left cliques.

First, consider part 1. Let A' be the right successor of A. By Fact 4, A' r-spans A. By Lemma 3.3, B must r-span A. By Lemma 3.5, the upper hole of A' is above that of B. It follows that the vertical separation between the upper holes of B and A is at most $l(A')$ and $l(A') \leq l(A)2^{\Delta}$, because A is not a right jumper. Part 1 follows.

Next, consider part 2. Let D be the smallest nonjumper strip in C, and let D' be the right successor of D (see Figure 12). D' exists because D is not a terminal. We will show in the next paragraph that the vertical separation between the upper holes of B and A is at most $l(D')$. Since D is not a right jumper, $l(D') \leq l(D)2^{\Delta}$. Further, since there are at least $\gamma\Delta$ strips smaller than A in C, using the increase in length categories given by Lemma 3.4, we get $l(D') \leq l(D)2^{\Delta} \leq \frac{l(B)}{2^{(\gamma)\Delta}}2^{\Delta} = \frac{l(B)}{2^{(\gamma-1)\Delta}}$, as required.

It remains to show that the vertical separation between the upper holes of B and A is at most $l(D')$. By Lemma 3.3, A and B both r-span D. By Lemma 3.5, A and B are between D and D'. Since D' is the right successor of D, D' must r-span D (see Fact 4); therefore, the lower hole of D' is below the upper hole of A. Further, by Lemma 3.5, the upper hole of D' is above that of B. It follows that the vertical separation between the upper holes of B and A is at most $l(D')$. □

Remark. Note that the above proof directly used only Fact 4 among the six facts listed above, and Lemmas 3.3, 3.4, and 3.5.

Proof of Lemma 3.7. We prove the lemma for a right clique C; the argument for the left is analogous.

By Lemma 3.3, B r-spans A. Let A' be the right successor of A and B' that of B. By Fact 4, A' r-spans A. By Lemma 3.5, B and B' are between A and A', A' cannot r-span B, and the upper hole of A' is above that of B. It follows that the right blocking hole d of B must be to the left of A' (see Figure 13) and d can be only in the two hatched regions in the figure. But if it is in the upper of these two regions, by Fact 5, the right witness cells for A and B are independent. Therefore, it must be in the lower hatched region, which is completely below A. □

Remark. Note that the above proof directly uses only Facts 4 and 5 among the six facts listed above, and Lemmas 3.3 and 3.5.

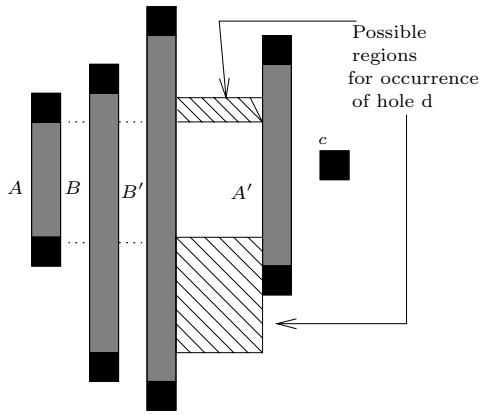


FIG. 13. *Possible locations of the blocking hole of B.*

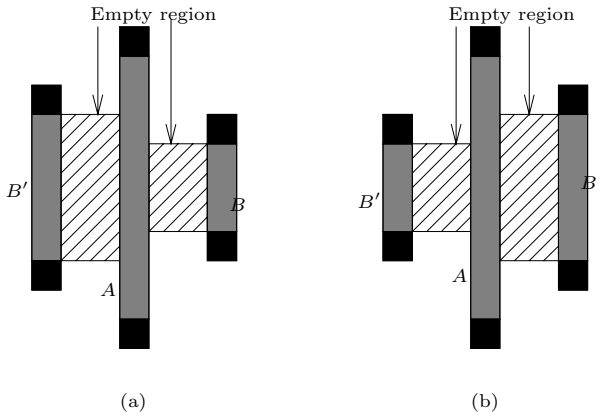


FIG. 14. *Possibilities precluded by Lemma* 3.8: (a) *B′ l-spans B.* (b) *B r-spans B′.*

Proof of Lemma 3.8. Suppose B' l-spans B (Figure 14(a)). The hatched region must be hole-free since A l-spans B and r-spans B' (see Lemma 3.3). This means that B has its left blocking hole to the left of B'. Next, B's left successor is either to the left of B' or to the right of A. This is true because any strip between B' and A which l-spans B must r-span B' as well and cannot have its left blocking hole to the left of B' (recall from Facts 1 and 3 that B and its left successor must have the same left blocking hole). Using Facts 4 and 5, it follows that the left witness cell of B is independent from that of A, a contradiction. Therefore B' cannot l-span B.

By an argument symmetric (see Figure 14(b)) to the one above, B cannot r-span B'. □

Remark. Note that the above proof directly used only Facts 1, 3, 4, and 5 among the six facts listed above, and Lemma 3.3.

4. The general case: Removing ill-behaved strips. We now need to handle the general case. Assumption 2 is not very hard to handle; it is possible to show that a good fraction of strips define witnesses on at least one of the two sides. Assumption 1 is the most severe: strips of arbitrary lengths could result in large families and cliques in which the progression of strip lengths described by Lemma 3.4 is absent. This

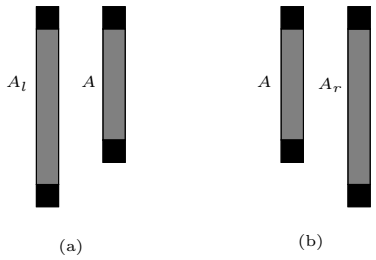


FIG. 15. *The two kinds of strips considered in Lemma* 4.1.

forces us to alter the notion of successor/family defined in this section and partition strips into classes based on the nature of left and right successors. A detailed analysis is then needed for each of these classes.

Recall that we have already identified and decided to ignore unnecessary strips. The aim now is to discard some more ill-behaved strips and then redefine successors and families for the strips which remain. These remaining strips will constitute a constant fraction of the number of the rectangles laid out by our algorithm, and each such strip will have the property that it defines either a left witness cell or a right witness cell or both. Unlike the previous special case, it will no longer be true that each strip is part of some family. Each strip will still have a successor, though.

4.1. Discarding strips. Consider two sets of strips. The first set comprises necessary strips A with the property that A_l exists, where A_l is the closest necessary strip to the left, if any, such that the upper holes of A_l and A are horizontally aligned (Figure 15(a)) and A_l l-spans A. The second set comprises necessary strips A with the property that A_r exists, where A_r is the closest necessary strip to the right, if any, such that the upper hole of A_r and A are horizontally aligned (Figure 15(b)) and A_r r-spans A. By Lemma 4.1 below, the strips in the smaller of the above two sets can be ignored. Without loss of generality, assume that the former set is smaller. Let $\mathcal{S}'$ denote the set comprising the remaining strips. For all strips A in $\mathcal{S}'$, A_l does not exist. This property, along with the subsequent definition of successors, will ensure that left witnesses are always defined for all strips in $\mathcal{S}'$.

LEMMA 4.1. *One of the above two sets must have size at most half the number of necessary strips.*

Proof. This follows because if A is in the first set then A_l is not in the second. □

Defining categories, doubling strips and nondoubling strips. We classify strips in $\mathcal{S}'$ into categories based on length. All strips with length in the range $[2^i, 2^{i+1})$ are in the ith category, $i \leq \log n - 1$.

Consider a strip A in $\mathcal{S}'$. Let B be the closest strip to the right of A in $\mathcal{S}'$, if any, which r-spans A and is in the same category as A. If B exists, then A is said be *right nondoubling*, and B is said to be the *right successor* of A. In this case, any strip in $\mathcal{S}'$ to the right of A and to the left of B which r-spans A must be in a higher category than A and therefore has its right blocking hole vertically aligned with or to the left of B. All other strips A are called *right doubling* strips. Analogous notions are defined to the left. As will be shown in Lemma 4.5 shortly, each strip is the right successor of at most one right nondoubling strip.

Note that we have not yet defined successors for right/left doubling strips. We will do so after we redefine families later in this section.

Remark. The strips considered in sections 3 and 3.1 are right and left doubling because of Assumption 1. As remarked in section 3.1, the notion of successor defined there is different from that defined above. The successor defined earlier corresponds to the successor of a right or left doubling strip.

LEMMA 4.2. *The number of strips in $\mathcal{S}'$ which are both right nondoubling and left nondoubling is at most $|\mathcal{S}'|/2$.*

Proof. Consider strip A, which is both right nondoubling and left nondoubling. Let B be its right successor. Note that no other right nondoubling strip has B as a right successor (this will be shown formally in Lemma 4.5). We show in the next paragraph that B cannot be a left nondoubling strip. It follows that for each A which is both right nondoubling and left nondoubling, there is a unique B which is left doubling. The lemma follows.

Suppose B is left nondoubling. Let D denote its left successor. Then D is between A and B and l-spans B. Since B r-spans A, D r-spans A as well. Since A, B, D are all in the same category, D would be the right successor of A, a contradiction. □

Let $\mathcal{S}$ denote the subset of $\mathcal{S}'$ comprising strips which are either left doubling or right doubling or both. The following lemma shows that $\mathcal{S}$ contains at least one-fourth of all the necessary strips. All further references to strips in the paper will be to the strips in $\mathcal{S}$. We will account for only these strips; the remaining strips are discarded.

LEMMA 4.3. *The number of strips in $\mathcal{S}$ is at least one-fourth the number of rectangles laid out to cover the polygon.*

Proof. Three kinds of strips have been ignored so far in defining the set $\mathcal{S}$:

1. unnecessary strips;
2. strips A with the property that A_l exists, where A_l is the closest necessary strip to the left such that the upper holes of A_l and A are horizontally aligned (Figure 15(a)) and A_l l-spans A;
3. strips A, which are both left and right nondoubling.

Unnecessary strips have the same associated rectangle as some necessary strip. By Lemma 4.1, the number of strips of the second type is at most half the total number of necessary strips. Finally, by Lemma 4.2, the number of strips of the last type is at most half of the remainder obtained by removing strips of the first two types. □

4.2. Defining families and witnesses. A crucial difference from before is that right families comprise only right doubling strips, and similarly for left families.

A *right family* is defined to be a set of right doubling strips in $\mathcal{S}$ with the same right blocking hole. The right successor of a right doubling strip is the next strip A' to the right in the family. The rightmost strip in a right family has no successor, and is called a right terminal strip, as before. Left families and successors are defined similarly. Notice that this is the same definition as in section 3.1, where all strips were left and right doubling.

We have now defined successors for all strips, whether doubling or not. As before, the right (left, respectively) witness cell of a nonterminal strip A (whether doubling or nondoubling) is the cell in its right (left, respectively) successor, horizontally aligned with the upper hole of A. This witness is not defined if the upper holes of A and its successor are horizontally aligned. The right (left, respectively) witness cell of a right (left, respectively) terminal strip A is defined as before; i.e., it is the cell in A which is horizontally aligned with its right (left, respectively) blocking hole.

The next lemma shows that each strip in $\mathcal{S}$ defines a left witness and is a straightforward consequence of the definition of $\mathcal{S}'$.

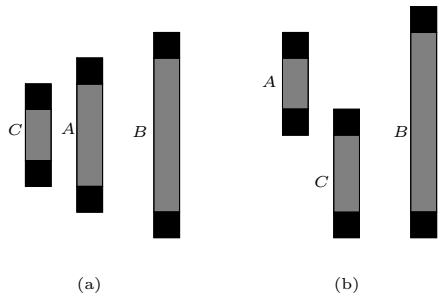


FIG. 16. *Two possible configurations of strips A, B, C, where B is the right nondoubling successor of both A and C.*

LEMMA 4.4. *For all nonterminal strips $A \in \mathcal{S}$, the upper hole of A is not horizontally aligned with the upper hole of its left successor. Therefore, each strip A defines a left witness cell and possibly a right witness cell.*

LEMMA 4.5. *Any strip is the right (left, respectively) successor of at most two strips, one right (left, respectively) nondoubling and one right (left, respectively) doubling.*

Proof. We prove the "right" case; the "left" case is analogous. First, suppose there are two right nondoubling strips A, C whose right successor is strip B. Then A, B, C belong to the same category, and hence $l(A) < l(B) < 2l(A)$ and $l(C) < l(B) < 2l(C)$. B must r-span both A and C. This can happen only either if A r-spans C (or vice versa) or A and C are both disjoint (see Figure 16). The former possibility cannot arise, since in that case the right successor of C would be A (or vice versa), and in the latter case, $l(B) \geq l(A) + l(C) \geq 2\min\{l(A), l(C)\}$, which implies that B cannot be in the same category as either A or C, a contradiction.

Next, consider right doubling strips. If the right successor of a right doubling strip A is strip B, then A and B must belong to the same right family and B is the first strip to the right of A in its family. So B cannot be the right successor of any other right doubling strip. ∎

4.3. Classifying strips. Based on the above, we can classify strips in $\mathcal{S}$ as follows. Recall that strips in $\mathcal{S}$ are left doubling or right doubling or both.

Class 1 This class contains all strips which are either terminal strips or jumper strips. This class has two subclasses.
- *Class* 1.1 This class contains all strips which are either right terminals strips or left terminal strips.
- *Class* 1.2 This class contains all strips which are either right jumper strips or left jumper strips.

Class 2 This class includes all strips not in Class 1 and which define witness cells in both directions. This class has three subclasses.
- *Class* 2.1 This class contains strips which are left doubling and right doubling.
- *Class* 2.2 This class contains strips which are left doubling and right nondoubling.
- *Class* 2.3 This class contains strips which are left nondoubling and right doubling.

Class 3 This class includes all strips not in Class 1 and which define only left witness cells. This class has two subclasses.
- *Class* 3.1 This class contains strips which are left doubling.
- *Class* 3.2 This class contains strips which are left nondoubling.

The aim now is to bound the number of strips in each class by $O(|OPT|(\Delta + \frac{\log n}{\Delta}))$.

4.4. Properties of cliques: The general case. This section deals with the properties of cliques in the general case. We will generalize all the lemmas stated in section 3.2. We partition the optimal clique cover, OPT, into disjoint cliques in the same way as described in section 2.4.

DEFINITIONS. Define $T_r(C, i)$ to be the set of strips of class i, i being one of 1.1, 1.2, 2.1, 2.2, 2.3, 3.1 or 3.2, whose right witness cells are in clique C of OPT. $T_l(C, i)$ for left witness cells is defined similarly. As mentioned earlier, we will sometimes abuse notation and identify a strip with its witness cell.

Before continuing, we reiterate that all right families comprise right doubling strips only, and, similarly, left families comprise left doubling strips only. Note that with our new definition of successors and families for strips in $\mathcal{S}$, left versions (right versions, respectively) of Facts 1–6 of section 3.4 continue to hold for left doubling (right doubling, respectively) strips. This will allow us to use the proofs in section 3.4 here as well for these strips. In addition, it is clear that Facts 4 and 5 continue to hold for all cliques (and not just cliques associated with doubling strips). Therefore, the lemmas in section 3.2 which use only these facts will generalize to both the doubling and the nondoubling cases directly.

The following lemma is exactly Lemma 3.1 stated for the new definition of terminal strips. The proof is exactly the same as Facts 1, 2, and 6 continue to hold for doubling strips.

LEMMA 4.6. *All right (left, respectively) witness cells associated with right (left, respectively) terminal strips are independent. Therefore, the number of terminal strips is* $O(|OPT|)$.

The following lemma is exactly Lemma 3.2, stated in terms of the classes defined above. Again, the proof is exactly the same, because Facts 1, 3, and 5 continue to hold for doubling strips.

LEMMA 4.7. *All strips in* $T_r(C, i)$ *(*$T_l(C, i)$*, respectively) belong to distinct right families (left families, respectively) for all cliques* C *in* OPT *and* i *being one of* $2.1, 2.3$ *(*$2.1, 2.2, 3.1$*, respectively).*

The following lemma, a generalization of Lemma 3.3, proves nested structure for cliques (modulo exceptions).

LEMMA 4.8. *For any clique* C *in* OPT *and any class* i*,* i *being one of* $2.1, 2.2, 2.3$ *(*$2.1, 2.2, 2.3, 3.1, 3.2$*, respectively), there exists at most one strip (called the* exception*) whose removal makes* $T_r(C, i)$ *(*$T_l(C, i)$*, respectively) right nested (left nested, respectively).*

Proof. The proof for $T_l(C, i)$, i being one of 2.1, 2.2, 3.1, and for $T_r(C, i)$, i being one of 2.1, 2.3, is described in Lemma 3.3 (the same proof holds because Facts 1, 3, 4, and 5 and Lemma 4.7 continue to hold for doubling strips).

We shall consider the remaining case for $T_r(C, i)$ here, involving nondoubling cliques (i.e., i being 2.2). A similar proof will hold for $T_l(C, i)$ with i being one of 2.3, 3.2. We cannot appeal to the proof of Lemma 3.3 directly because Fact 3 does not hold for nondoubling strips.

However, it is easy to see that the fact stated in the beginning of the proof of Lemma 3.3 still holds. We will give the rest of the proof, assuming this fact is true.

Suppose the strips in $T_r(C, i)$ are not right nested. By the above fact, if each strip in $T_r(C, i)$ r-spans the smallest strip in $T_r(C, i)$, then there are no disjoint strips, and the strips in $T_r(C, i)$ must be right nested. So there must exist a strip in $T_r(C, i)$ which is disjoint from the smallest strip A in $T_r(C, i)$; consider the smallest such strip B. Clearly, neither A nor B can r-span any strip in $T_r(C, i)$. We will show that all

other strips D in $T_r(C,i)$ must r-span the lower of A,B. It would then follow from the above fact that the strips in $T_r(C,i)$ with the upper of A,B removed are right nested. We consider the case when B is below A; the other case is identical.

Consider a strip D as above, and suppose it does not r-span B. By the fact above, it must be disjoint from B; in addition, it either r-spans A or is disjoint from A as well. Let X,Y denote the upper two strips among A,B,D. So either X and Y are disjoint (as in Figure 10(b)) or (without loss of generality) $l(X) < l(Y)$ and Y r-spans X (as in Figure 10(a)). Let X',Y' be the right successors of X,Y, respectively. Since the right witnesses of A,B,D form a clique, rectangle R containing these witnesses has its lower edge below the lower holes of X,Y and upper edge above the upper holes of X,Y. This, coupled with the fact that X' must stab vertically through R, implies that $l(X') > l(Y), l(X)$; further, if X,Y are indeed disjoint, then $l(X') > l(X)+l(Y) > 2l(X)$. But the latter cannot happen as X is right nondoubling, and therefore X' and X are in the same category, which implies that $l(X') < 2l(X)$. Thus, it must be the case that Y and X are not disjoint; i.e., Y r-spans X. Then $l(X) < l(Y) \leq l(X')$, and Y must be in the same category as X. Since Y r-spans X, is in the same category as X, and is to the left of X', X' cannot be the right successor of X, a contradiction. □

Remark. As we mentioned earlier in section 3.2, we can now ignore exception strips from all cliques of OPT.

Next, we generalize Lemma 3.4.

LEMMA 4.9. *Strips in $T_l(C,i)$, i being one of* $2.1, 2.2, 2.3, 3.1, 3.2$*, and in $T_r(C,i)$, i being one of* $2.1, 2.2, 2.3$*, are in distinct size categories.*

Further, consider a right family (left family, respectively) or a set $T_r(C,i)$ of strips, i being one of $2.1, 2.2, 2.3$ *(set $T_l(C,i)$ of strips, respectively, i being one of* $2.1, 2.2, 2.3, 3.1, 3.2$*) for some clique C in OPT. For any x, the number of strips whose length is more than 2^x times the length of one of the two previous strips to the left (right, respectively) is $O(\frac{\log n}{x})$.*

Proof. For the first part, we cannot use the proof of the first part of Lemma 3.4, because that proof was based on Assumption 1, and so we describe it below.

Suppose two strips A,B in $T_l(C,i)$, i being one of 2.1, 2.2, 2.3, 3.1, 3.2, are in the same size category. Without loss of generality, assume B is to the left of A. By Lemma 4.8, B l-spans A. By the definition of a left nondoubling strip, A must be left nondoubling, and its left successor must either lie to the right of B or be B itself. In either case, A's left witness cell will be independent from B's left witness cell, a contradiction.

A similar proof holds for two strips A,B in $T_r(C,i)$, i being one of 2.1, 2.2, 2.3.

The second part of the lemma follows by using the same argument as in Lemma 3.4, since this uses only Fact 1 for families, and the first part for cliques. □

The following lemma generalizes Lemma 3.5.

LEMMA 4.10. *Let $A,B \in T_r(C,i)$ ($T_l(C,i)$, respectively) for some clique C in OPT and some class i, i being one of* $2.1, 2.2, 2.3$ *(*$2.1, 2.2, 2.3, 3.1, 3.2$*, respectively), with $l(A) < l(B)$. Let A',B' be the right successors (left successors, respectively) of A,B, respectively. These four strips must be in the following order from left to right (right to left, respectively): A,B,B',A'. In addition, A' cannot r-span (l-span, respectively) B and must have its upper hole above that of B.*

Proof. The proof for $T_l(C,i)$, i being one of 2.1, 2.2, 3.1, and for $T_r(C,i)$, i being one of 2.1, 2.3, is described in Lemma 3.5 (the same proof holds because Facts 1, 3, 4, and 5 continue to hold for doubling strips).

We need to prove the lemma for the remaining cases only. We prove the lemma for $T_r(C, 2.2)$; the other cases have analogous proofs.

From Lemma 4.8, it follows that B r-spans A and hence must lie to the right of A. If B is to the right of A', then the right witness cells a and b of A and B, respectively, are independent (see Figure 11(b)). Therefore, B is to the left of A' and to the right of A. To show that the right successor of B also lies between A and A', we show in the next paragraph that A' cannot r-span B. Since A' and B both r-span A, they cannot be disjoint either. Then it follows that the right blocking hole of B is vertically aligned with or to the left of A'. Therefore, the right successor of B is also to the left of A'.

Suppose A' r-spans B. Since we are considering category 2.2, A' and A are in the same category. Since B r-spans A and A' r-spans B, A, B are also in the same category, and then B, and not A', will be the right successor of A.

It remains to show that the upper hole of A' is above that of B. Recall that B's right witness cell lies on the horizontal line containing the upper hole of B. If the upper hole of A' is horizontally aligned with or below that of B, the witness cells a and b of A and B, respectively, would be independent (see Figure 11(d)). Hence, the upper hole of A' lies above that of B. □

DEFINITIONS. As before, we define strip A to be a *right jumper* if its right successor has length at least $2^{\Delta} l(A)$. Since Δ will be set to at least 1, all right jumpers are actually right doubling strips. Left jumpers are defined analogously.

The following lemma generalizes Lemma 3.6.

LEMMA 4.11. *Consider strips $A, B \in T_r(C, i)$ ($T_l(C, i)$, respectively) for some clique C in OPT and i being one of* $2.1, 2.2, 2.3$ *(*$2.1, 2.2, 2.3, 3.1, 3.2$*, respectively). Suppose $l(A) < l(B)$. The following two facts hold.*

1. *The vertical separation between the upper holes of A and B is at most $2^{\Delta} l(A)$.*
2. *If A is not amongst the smallest $\gamma\Delta$ nonjumper strips in $T_r(C, i)$ ($T_l(C, i)$, respectively), the vertical separation between the upper holes of A and B is at most $\frac{l(B)}{2^{(\gamma-1)\Delta}}$.*

Proof. The proof is identical to that of Lemma 3.6 (the appropriate generalizations of the lemmas used there have to be invoked) because that proof uses only Fact 4 and Lemmas 4.8, 4.9, and 4.10, which hold for both doubling and nondoubling strips. □

We generalize Lemma 3.7 next.

LEMMA 4.12. *Let $A, B \in T_r(C, i)$ ($T_l(C, i)$, respectively) for some clique C in OPT and i being one of* $2.1, 2.2, 2.3$ *(one of* $2.1, 2.2, 2.3, 3.1, 3.2$*, respectively). Further, suppose $l(A) < l(B)$. Then A lies completely above the right (left, respectively) blocking hole of B.*

Proof. The proof is the same as that of Lemma 3.7 (again, the appropriate generalizations of the lemmas used there have to be invoked), as that proof uses only Facts 4 and 5 and Lemmas 4.8 and 4.10, which continue to hold for both doubling and nondoubling strips. □

Finally, we need the following lemma, which generalizes Lemma 3.8.

LEMMA 4.13. *Let C and C' be the cliques in OPT containing the right and left witness cells, respectively, of a strip A. Let B be a strip in $T_r(C, i)$ smaller than A. Let B' be strip in $T_l(C', i)$ smaller than A. If i is one of* 2.1 *or* 2.2*, then B' cannot l-span B. If i is one of* 2.1 *or* 2.3*, then B cannot r-span B'.*

Proof. The proof for $i = 2.1, 2.2$ is identical to that of Lemma 3.8 as Facts 1, 3, 4, and 5 used in that proof continue to hold in the left direction for $i = 2.1, 2.2$ (these

involve left doubling strips). Note that the whole of the proof for Lemma 3.8 is not being invoked; rather, only the first half is being invoked. To invoke the second half (namely, B cannot r-span B') as well, we will need the above facts to hold in the right direction. This is indeed true for Class 2.1, which is right doubling as well but not for Class 2.2. Invoking this second half for Class 2.1, we get the last part of the lemma for this class. The proof for $i = 2.3$ is analogous. □

4.5. Accounting for strips. The strips in each class are accounted for separately. The number of strips in Class 1.1 is $O(|OPT|)$ by Lemma 4.6. The number of strips in Class 1.2 is $O(|OPT|\frac{\log n}{\Delta})$ by Lemmas 4.6 and 4.9.

Class 2.1 contains strips that are doubling in both directions, like the strips in the special case considered in previous sections. The number of strips in Class 2.1 can be shown to be $O(|OPT|(\Delta + \frac{\log n}{\Delta}))$ by a proof identical to those of Lemma 3.9 and Corollary 3.10; however, invocations of lemmas in section 3.2 need to be modified to point to their respective counterparts in section 4.4 (see the remark in section 3.3).

The reason why the proof of Lemma 3.9 does not extend to other classes as well is that Lemma 3.9 uses Lemma 3.8 in both directions, i.e., to claim that B' cannot r-span B and B cannot l-span B'; this can be done only for Class 2.1. Therefore, the remaining classes need separate proofs, which are given below.

4.5.1. Classes 2.2 and 2.3.

LEMMA 4.14. *Consider a strip $P \in T_r(C, 2.2)$ and its right follower $Q \in T_r(C, 2.2)$ (Q need not be defined). Then one of the following must hold.*

1. *P is either the largest strip or among the smallest $\Delta + 2$ strips in $T_r(C, 2.2)$.*
2. *Q is among the smallest $4\Delta + 1$ strips in $T_l(C', 2.2)$, where C' denotes the clique in OPT containing the left witness cell of Q.*
3. *Let P' be the strip in $T_r(C, 2.2)$ which is to the left of P and shorter than P such that the number of strips longer than P' and shorter than P in $T_r(C, 2.2)$ is 3. Then either P' does not exist or $2^{3\Delta} l(P') \leq l(Q)$.*
4. *Let Q' denote the strip in $T_l(C', 2.2)$ whose left follower is Q. Then $2^{\Delta} l(Q') \leq l(Q)$.*

Proof. We suppose that none of the four conditions hold and derive a contradiction.

Since condition 1 does not hold, Q exists. The proof then proceeds using the following claims, which are proved in sebsequent paragraphs. We claim that any strip in $T_r(C, 2.2)$ with length less than $l(Q')/2$ must lie completely above Q'. Since condition 3 is violated, P' must exist. We then show that P' has length less than $l(Q')/2$ and therefore lies above Q'. Since Q r-spans P' by Lemma 4.8, the vertical separation between the upper holes of Q and Q' is at least $l(P')$. By the violation of condition 2 and Lemma 4.11 applied to Q' and Q, the vertical separation between the upper holes of Q and Q' is at most $\frac{l(Q)}{2^{3\Delta}}$. It follows that $l(P') \leq \frac{l(Q)}{2^{3\Delta}}$. This satisfies condition 3, a contradiction.

First, we show that any strip R in $T_r(C, 2.2)$ with length less than $l(Q')/2$ must lie completely above Q'. Clearly, R lies to the left of Q. Its right successor S is to the right of Q, by Lemma 4.10. Further, S has size less than $l(Q')$ (since $l(R) < l(Q')/2$ and Class 2.2 is right nondoubling). In addition, by Lemma 4.10, the upper hole of S is above that of Q. It follows that the lower hole of S must also be above that of Q'; otherwise, $l(S) \geq l(Q')$, a contradiction. Suppose that R is not completely above Q'. We will get a contradiction as follows. Since R is not completely above Q' and S r-spans R, S is not completely above Q' either. Then one of the two situations shown

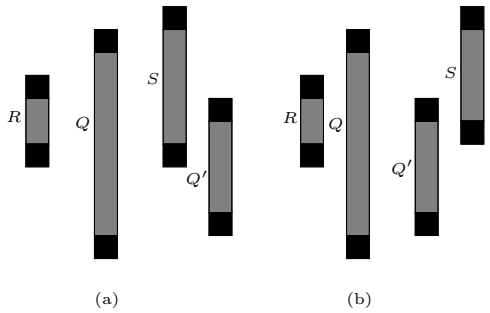


FIG. 17. (a) *S is to the left of Q', and Q does not l-span Q'.* (b) *S is to the right of Q', and S does not r-span R since it is not above Q'.*

in Figure 17 must hold, depending upon whether S is to the left or right of Q'. In the first case, Q cannot l-span Q', and in the second case S cannot r-span R, both contradictions. Therefore, R is completely above Q'.

Second, we show that P' has length less than $l(Q')/2$. To do this, we will show that $l(P) \leq 2l(Q')$. Then, since there are three strips between P and P' in $T_r(C, 2.2)$, $l(P') < l(P)/4 \leq l(Q')/2$. That $l(P) \leq 2l(Q')$ is shown as follows. We show in the next paragraph that P must have its upper hole aligned with or below that of Q and its lower hole aligned with or above that of Q'. Thus $l(P)$ is at most the vertical separation between the upper hole of Q and the lower hole of Q'. Since condition 2 is violated, Lemma 4.11 applied to Q' and Q implies that the vertical separation between their upper holes is at most $\frac{l(Q)}{2^{3\Delta}}$. Thus the vertical distance between the upper hole of Q and the lower hole of Q' is at most $l(Q') + \frac{l(Q)}{2^{3\Delta}} < l(Q')(1 + \frac{1}{2^{2\Delta}})$, by the violation of condition 4. Thus $l(P) \leq l(Q')(1 + \frac{1}{2^{2\Delta}}) \leq 2l(Q')$, as required.

It remains to show that P must have its upper hole aligned with or below that of Q and its lower hole aligned with or above that of Q'. By Lemma 4.8, Q r-spans P, and therefore the upper hole of P is aligned with or below that of Q. By the violation of condition 1 and by Lemma 4.8, which states that all strips in $T_r(C, 2.2)$ are in distinct categories, there exists a strip R in $T_r(C, 2.2)$ such that $l(R) < \frac{l(P)}{2^{\Delta+1}} < \frac{l(Q)}{2^{\Delta+1}}$. Then, by the violation of condition 4, $l(R) < l(Q')/2$. From the earlier part of this proof, it follows that R lies completely above Q'. Clearly, R is to the left of P. By Lemma 4.8, P r-spans R, and therefore the upper hole of P is above that of Q'. Further, by Lemma 4.13, P cannot l-span Q'. Therefore, the lower hole of P must be aligned with or above that of Q'. □

COROLLARY 4.15. *The number of strips in Class* 2.2 *is* $O(|OPT| * (\frac{\log n}{\Delta} + \Delta))$.

Proof. We consider four subclasses, depending upon which of the conditions in Lemma 4.14 is satisfied. The number of strips P which satisfy the first condition is clearly $O(|OPT| * \Delta)$ because each clique in OPT has $O(\Delta)$ such strips. The number of strips P which satisfy the third condition is $O(|OPT| * \frac{\log n}{\Delta})$, using arguments similar to those used for conditions 4 and 5 of Lemma 3.9 in the proof of Corollary 3.10. Next, consider strips P such that either condition 2 or 4 holds. Such a strip P has a unique right follower Q in $T_r(C, 2.2)$. Note that any strip in $T_r(C, 2.2)$ is the right follower of at most one strip. Thus it suffices to bound the number of strips Q which are right followers of strips P satisfying condition 2 or 4. Using arguments similar to those in Corollary 3.10, the number of such strips Q satisfying condition 2 can be shown to be $O(|OPT| * \Delta)$, and the number of such strips Q satisfying

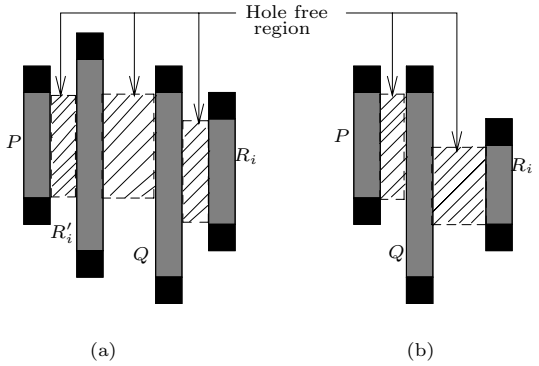


FIG. 18. *The two scenarios in Lemma* 4.17: (a) $j = 2.3, 3.2$. (b) $j = 2.1, 2.2, 3.1$.

condition 4 can be shown to be $O(|OPT| * \frac{\log n}{\Delta})$. □

A similar argument as above works for Class 2.3.

COROLLARY 4.16. *The number of strips in Class* 2.3 *is* $O(|OPT| * (\frac{\log n}{\Delta} + \Delta))$.

4.5.2. Classes 3.1 and 3.2.

LEMMA 4.17. *Consider a strip* P *in* $T_l(C, 3.1)$ *or* $T_l(C, 3.2)$. *Let* Q *be the right successor of* P. *Let* j *be the class containing strip* Q *and* C' *be the clique containing the left witness cell of* Q. *Then either* $j = 1.1$ *or* $j = 1.2$, *or* Q *is among the smallest* $2\Delta + 2$ *strips in* $T_l(C', j)$.

Proof. Since P belongs to Class 3.1 or Class 3.2 and not to Class 1.1, it has a well-defined right successor Q. Further, since P does not define a right witness cell, the upper holes of P and Q are horizontally aligned (see Figure 18). In addition, if P is right nondoubling, then $l(Q) < 2l(P)$, and if P is right doubling, then $l(Q) \leq l(P)2^{\Delta}$, as P is not a right jumper (i.e., it is not in Class 1.2). Note that if P is in Class 3.2, then it must be right doubling, as all strips in $\mathcal{S}$ are either left doubling or right doubling or both (recall the definition of $\mathcal{S}$ from section 4). We consider various cases depending upon the nature of Q.

Suppose $j \neq 1.1$ and $j \neq 1.2$. Then Q is not a left terminal strip. Let $T_l(C', j)$ have k strips smaller than Q. Let these be $R_1, \ldots, R_k$, in increasing order of length. We need to show that $k \leq 2\Delta + 1$.

By Lemmas 4.8 and 4.9, the R_is and Q together form a left nested set of strips and therefore belong to distinct categories. Note that since $j \neq 1.1$, Q and each of the R_is have left successors. Let R'_i denote the left successor of R_i. There are two cases now, depending upon whether j is one of 2.3, 3.2 or one of 2.1, 2.2, 3.1.

First, suppose j is one of 2.3, 3.2. Then Q and the R_is are all left nondoubling (see Figure 18(a)). Then R'_i is in the same category as R_i. Since all R_is and Q are left nested and in distinct size categories, $l(R'_1) < l(R'_2) < \cdots < l(R'_k) < l(Q)$, and $2^{k-1}l(R'_1) < l(Q)$. All R'_is must be between P and Q. For R'_i cannot be to the right of Q by Lemma 4.10. And, if R'_i is to the left of P, then the left witness cells of Q and R_i are independent because Q's left witness cell is on the horizontal line joining the upper holes of P and Q. From Lemma 4.10, the upper hole of each R'_i is above the upper holes of both Q and P. Each R'_i must r-span P because R'_i must l-span R_i and the hatched region is hole-free (because Q r-spans P). Thus $l(P) < l(R'_1)$. Therefore $l(Q) > 2^{k-1}l(R'_1) > 2^{k-1}l(P)$. So if $k \geq \Delta + 1$, $l(Q) > l(P)2^{\Delta}$, a contradiction (see the first paragraph of this proof; note that $\Delta \geq 1$). It follows that $k \leq \Delta$ in this case.

Second, suppose j is one of 2.1, 2.2, 3.1. Q and the R_is are all left doubling (see Figure 18(b)). Note that the hatched regions in the figure must be hole-free, as Q must r-span P and l-span R_i. Then the lower hole of each R_i must be below that of P; otherwise, the left witness cell for R_i will be on or to the left of P and independent from the left witness cell for Q, a contradiction. Therefore, the vertical distance between the upper hole of P (or of Q) and the lower hole of R_1 is at least $l(P)$. Since the R_is and Q belong to distinct categories, $2^{\Delta}l(P) \geq l(Q) > 2^{k-1}l(R_1)$ (the first inequality follows from the first paragraph of this proof). It follows that the vertical separation t between the upper holes of R_1 and Q is at least $l(P) - l(R_1) > (2^{k-1-\Delta} - 1)l(R_1)$. For $k \geq 2\Delta + 2$, $t > (2.2^{\Delta} - 1)l(R_1) > 2^{\Delta}l(R_1)$. This contradicts Lemma 4.11 (the first part, applied to R_1 and Q). Thus $k \leq 2\Delta + 1$ in this case, as required. □

COROLLARY 4.18. *The number of strips in Classes* 3.1 *and* 3.2 *is* $O(|OPT| * (\frac{\log n}{\Delta} + \Delta))$.

Proof. Each strip in these two classes has a right successor, which in turn has a left witness by Lemma 4.4. Further, by Lemma 4.5, any strip is the right successor of at most two strips. By Lemma 4.17, either (a) the right successor of a strip in these two classes is in Class 1.1 or 1.2, or (b) the right successor of a strip in these two classes is in some class $j \neq 1.1, 1.2$ and is amongst the smallest $2\Delta + 2$ strips in $T_l(C', j)$ for some clique C' in OPT. Strips in Classes 3.1 and 3.2 for which the right successor satisfies the latter property are clearly $O(|OPT| * \Delta)$ in number. And strips for which the right successor satisfies the former property are $O(|OPT|\frac{\log n}{\Delta})$ in number by Lemmas 4.6 and 4.9. □

4.5.3. Summing up.

THEOREM 4.19. *The number of rectangles needed to cover the given polygon is* $\Omega(\#N/\sqrt{\log n})$*, where* $\#N$ *is the number of necessary strips and therefore the number of rectangles used by our algorithm.*

Proof. From Corollaries 3.10, 4.15, 4.16, and 4.18, it follows that $|\mathcal{S}| = O(|OPT| * \max\{\frac{\log n}{\Delta}, \Delta\})$. By Lemma 4.3, $|\mathcal{S}|$ is at least a quarter of the number of rectangles used by our algorithm, which is equal to the number of necessary strips. The theorem follows by setting $\Delta = \sqrt{\log n}$. □

5. Counterexample. If the average family size was $O(1)$ for all polygons, then our claim that there always exist $\#F$ independent points will give a constant factor approximation algorithm. Unfortunately, this is not the case. Here we give an example of a polygon in which the average family size is $\theta(\frac{\log n}{\log \log n})$.

This example can also be slightly modified in a way such that all right witness points can be covered by $O(\frac{\#N \log \log n}{\log n})$ cliques and the average right clique size is $\Theta(\frac{\log n}{\log \log n})$, but the average family size (both left and right) remains $\Theta(\frac{\log n}{\log \log n})$. However, in this example, covering the left witness points requires $\Omega(\#N)$ cliques, and the maximum clique size is $O(1)$ for these points. This example was the key to our lower bound. We do not know whether there are examples where the average clique size is superconstant for both the left witness points and the right witness points. In this sense, our bound of $\sqrt{\log n}$ does not seem like an unnatural meeting point.

Let l be a parameter, which we will ultimately set to $\log n$. We give an example where the average left family size, average right family size, and average right clique size are all $\Theta(\log_l n)$.

Our polygon will have two kinds of holes, *nonblocking* and *blocking*. There will be $\Theta(n)$ blocking holes and $\Theta(n \log_l n)$ nonblocking holes; so most holes will be non-

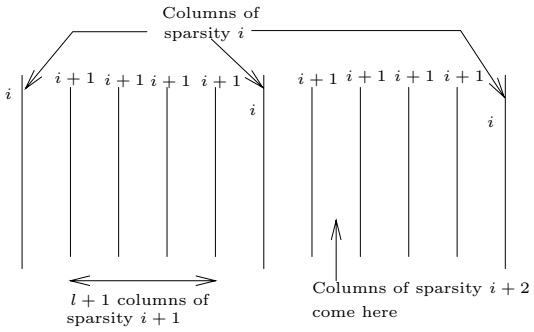


FIG. 19. *Arrangement of columns of sparsity i and $i+1$.*

blocking. First, we will describe the arrangement of nonblocking holes and then that of blocking holes. All subsequent references to strips will be to those formed by nonblocking holes.

Nonblocking holes are arranged in columns. Each column has a certain *sparsity*. A column with sparsity i will have $\frac{n}{l^i}+1$ holes in it, where $0 \le i \le \log_l n$; these holes will be put in rows $0, l^i, 2l^i, \ldots$. So the least sparse column will have $n+1$ holes in rows $0, 1, 2, 3, \ldots, n$, and the most sparse will have 2 holes in rows $0, n$. There will be $l^{i-1}(l+1)$ columns of sparsity i, $i \ge 1$. Therefore, the total number of nonblocking holes will be $\Theta(n \log_l n)$. The arrangement of these columns can be described by the following sequential procedure.

The leftmost and rightmost columns will have sparsity 0. Between these two columns, put $l+1$ columns of sparsity 1; these $l+1$ columns together constitute a *pack*. Then, between each pair of consecutive columns of sparsity 1, put a pack of $l+1$ columns of sparsity 2, and so on, as shown in Figure 19. Note here that a pack of sparsity $i+1$ columns is put only between pairs of consecutive columns of sparsity i which belong to the same pack; these sparsity i columns will not have any columns of sparsity less than i between them.

Blocking holes will always be placed as follows. First, we form right families comprising strips formed by nonblocking holes in columns which are not the last in their respective packs. Note that most (all but $\Theta(n)$) nonblocking holes lie in such columns. Consider a column C with sparsity i which is not the rightmost column in its pack. There are $\frac{n}{l^i}$ strips in such a column. These strips are organized into *groups* of l strips each, the strips in each group being vertically consecutive. Consider one such strip which is the jth strip in its group. We define a right successor s' for s, where s' is the unique strip in the column C' defined below which r-spans s; C' is the jth leftmost column amongst the pack of $l+1$ sparsity $i+1$ columns nested between C and the next sparsity i column to the right of C. The size of each right family defined by the above right successors is clearly large, i.e., $\Theta(\log_l n)$.

We will now arrange blocking holes so that all strips in each right family defined above will indeed have a common right blocking hole. For each right family defined above, put a blocking hole immediately to the right of the rightmost strip in such a way that it blocks all strips in this family. The number of blocking holes put is clearly $\Theta(n)$.

Thus, what we have achieved above is an arrangement of $\Theta(n \log_l n)$ holes where all but $O(n)$ strips lie in right families of size $\Theta(\log_l n)$. We remark here that all but $O(n)$ strips lie in left families of size $\Theta(\log_l n)$ as well in this arrangement. The picture

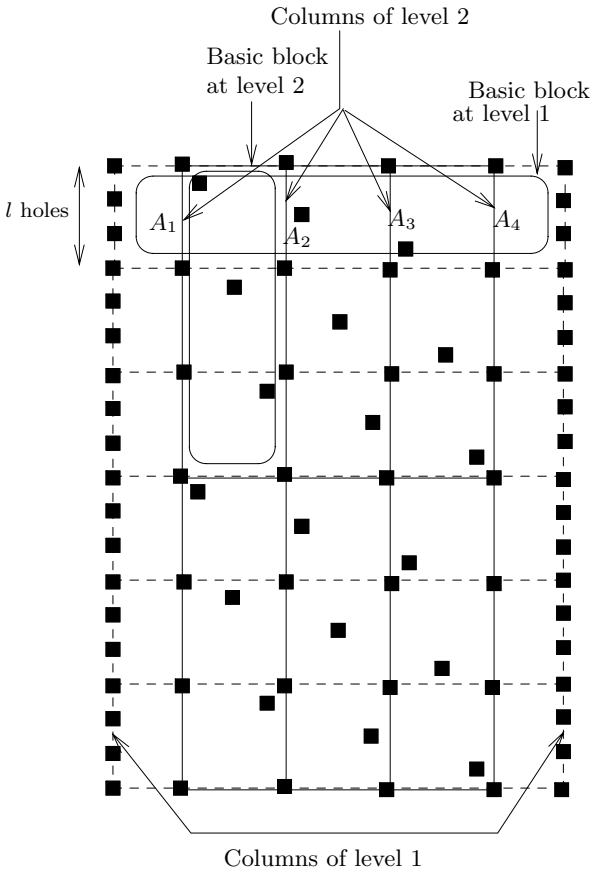


FIG. 20. *A basic block.*

of a "basic block" shown in Figure 20 will be helpful in seeing that this is true.

A basic block comprises the following holes.

1. holes bounding a group of strips on a column C of sparsity i, where C is not the rightmost column in its pack. Let h, h' be the topmost and bottommost such holes (see Figure 20);
2. all blocking holes whose vertical position is between h and h' and whose horizontal position is between C and the next column in the pack containing C;
3. all nonblocking holes which are located vertically between h and h' and are on columns of sparsity $i+1$ between C and the next column in the pack containing C.

5.1. Getting large right cliques. We need to make a modification to the above construction to get large right clique sizes while leaving left and right family sizes as before.

The modification is that columns containing nonblocking holes need to be shifted downwards by varying amounts while maintaining most (but not all) of the right and left families as such. This shifting is described by the following sequential procedure.

The shifting procedure is carried out in rounds. In the ith round, only columns of sparsity i or more will be shifted. Assume that the procedure has already been

executed for i rounds. At this point, for any column C of sparsity i, all columns of sparsity $i+1$ or more which appear between C and the next column in its pack will not have experienced any shift relative to C. For each strip s in a column C of sparsity i which is not the rightmost in its pack, let $map(s)$ denote that strip which has the same right blocking hole as s and lies on a column of sparsity $i+1$ between C and the next sparsity i column in C's pack. It can be verified from the shifting procedure below that $map(s)$ is always well defined and that a basic block (such as the one shown in Figure 21) has blocking holes for strips in C forming a group distributed diagonally (this basic block requires forming groups of l consecutive strips on column C, leaving the first $l-1$ strips out of this grouping; this is to account for the shifts made to C so far). So each basic block formed by groups on C looks like the one in Figure 21, except that holes in this basic block on sparsity $i+1$ columns between C and the next sparsity i column D in C's pack are all aligned with h, the top hole of this group. The $i+1$st round proceeds as follows.

For each column C of sparsity i which is not the rightmost column in its pack, consider any group of strips on C. Let h be the topmost hole in this group. Each strip s in this group is considered in turn. Let $\mathcal{P}$ denote the pack of sparsity $i+1$ strips nested between C and the next sparsity i strip to the right. The column C'' immediately preceding the column C' containing $map(s)$ in $\mathcal{P}$ is shifted down so that the hole which was horizontally aligned with h is now aligned with the upper hole of s (see Figure 21). In addition, all columns nested between C'' and C' will also be shifted down so that no relative shift is introduced between C'' and these columns in this round. For future reference, we denote the strip on C'' which now r-spans s by $cmap(s)$ and the strip on C' which now r-spans s by $newmap(s)$. Note that $cmap(s)$ is defined unless s is the first strip in its group. Figure 21 shows a basic block after this modification.

The above shifting procedure modifies right families, because right successors of strips could have changed. For each strip s in C, the right successor changes from $map(s)$ to $newmap(s)$. Families defined by this new definition of right successor are also large, essentially because a right successor can be defined for every strip other than those which are on the last columns in their packs. Thus, all but $O(n)$ of the strips will continue to be in right families of size $\Theta(\log_l n)$.

Also, the average right clique size is $\Theta(\log_l n)$. To see this, note that the right witness points of $s, cmap(s), cmap(cmap(s)), \ldots$ form a right clique and that $cmap(s)$ is defined for all those s which are not the first strips in their respective groups or in the last columns in their respective packs. Since there are only $O(n)$ strips s which are either the first strips in their respective groups or in the last columns in their respective packs, the total number of right cliques is $\Theta(n)$ and the average right clique size is $\Theta(\log_l n)$.

It now remains to show that left families continue to be large after the above modification. Consider a column C of sparsity i and the next column D to its right in its pack. As is clear from Figure 21, left successors can be defined in the pack $\mathcal{P}$ for each of the strips in D, except those strips which are either the first or last in their respective groups. Defining left successors recursively in this way ensures that the average left family size is also $\Theta(\log_l n)$.

5.2. Large left and right cliques? In the above example, it can be seen with some effort that all left cliques have size $O(1)$. We do not know whether this example can be modified so that the average left and right cliques sizes are both large.

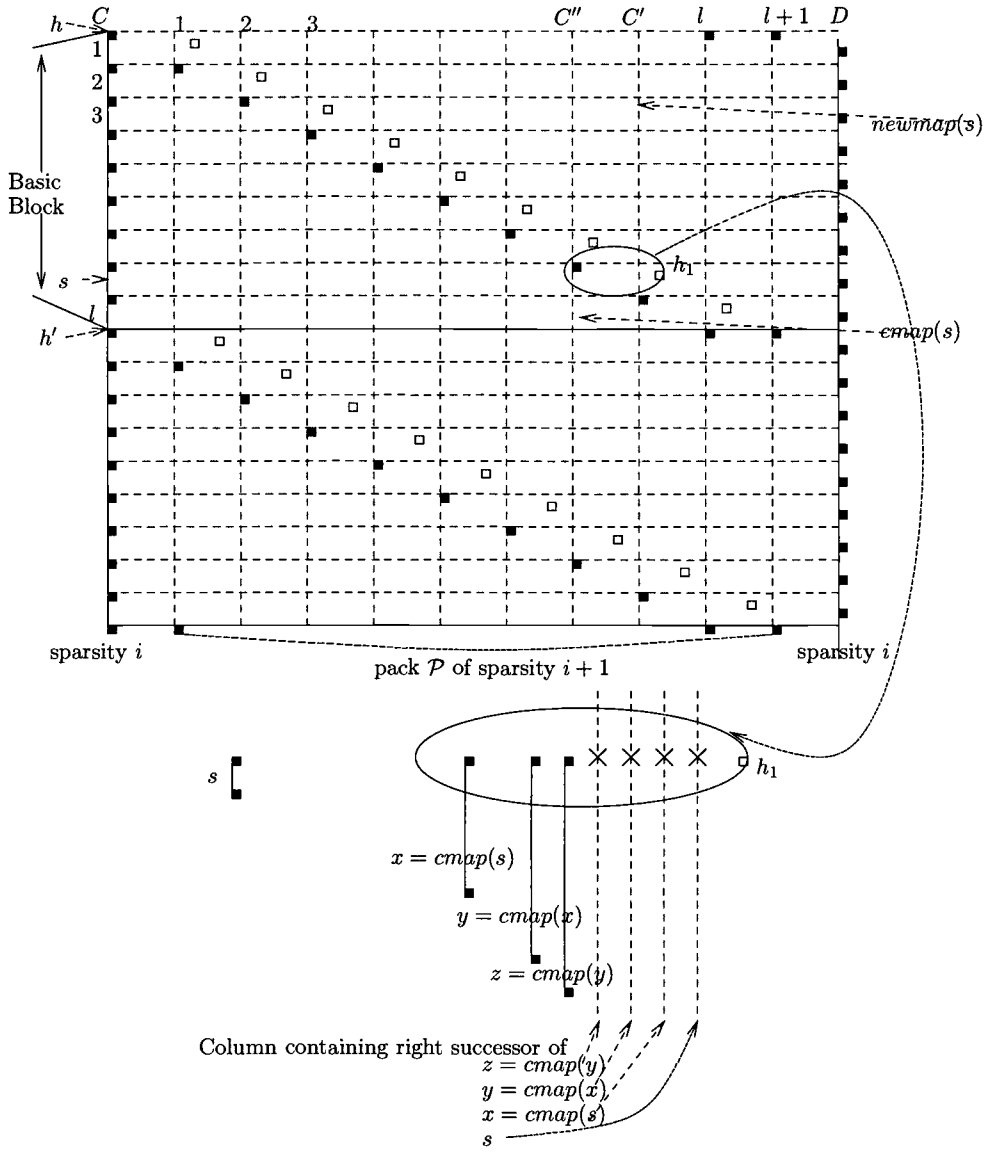


FIG. 21. *Two basic blocks obtained after the $i+1$st round of shifting. White holes are blocking holes. One large clique has been highlighted. The crosses are right witness points.*

6. Conclusions. A number of loose ends remain for this problem. The main question, of course, is whether the approximation factor can be brought down to $O(1)$. Another question is whether there exists a polygon whose clique cover and independent set numbers are small-o of the number of necessary strips.

Related problems. We briefly mention some related problems and the current state of knowledge on these problems.

Non–axis-parallel rectangles. One variant of the above rectangle covering problem is when the covering rectangles need not be axis-parallel.

Our techniques do not seem to extend to this case. However, they do extend

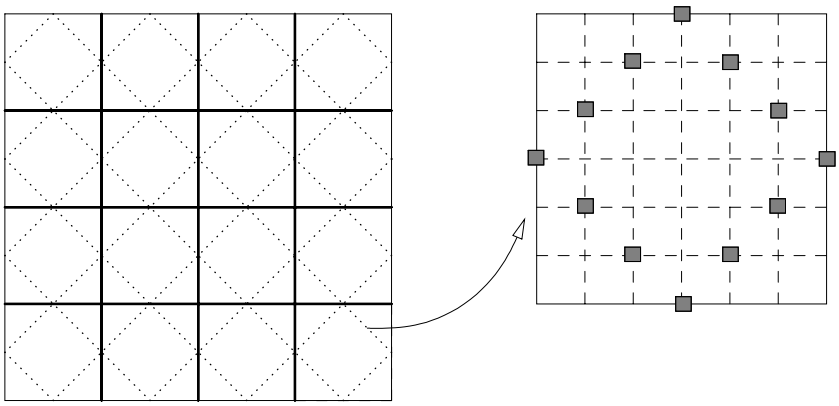


FIG. 22. *Covering with non–axis-parallel rectangles.*

even when all the covering rectangles must be inclined at the same angle or at one of a constant number of angles. But there are examples where rectangles inclined at an arbitrary number of angles are involved in the optimal cover. And, in this case, the issue seems to be different and related to the problem of covering a given set of points using a minimum number of straight lines. We do not know the exact nature of this relationship though. No $o(\log n)$ approximation factor is known for this problem either (see [1]). We describe an example below where the optimum cover has size $O(n)$ when non–axis-parallel rectangles are allowed, whereas it is $\Omega(n\sqrt{n})$ if only axis-parallel rectangles are allowed. This gives support for the intuition that the size of the minimum cover should be much smaller if non–axis-parallel rectangles are allowed.

Partition the $n \times n$ grid into a^2 tiles of size n^2/a^2 each (as in Figure 22(a)). One such tile is shown in Figure 22(b). Each tile has dimensions $n/a \times n/a$. The structure of the tiles results in a partition of the grid into triangles along the boundary and rhombuses inside. Each rhombus has n/a holes on each of its sides. First, consider the case when only axis-parallel rectangles are allowed. Each rhombus needs n/a axis-parallel rectangles for covering. Similarly, each triangle needs n/a rectangles to be covered. The total number of triangles is $4a$, and the total number of rhombuses is $\Theta(a^2)$. Therefore, the optimum has size $\Theta(a^2 n/a + 4an/a)$. Next, consider the case when arbitrarily oriented rectangles are allowed. Now each rhombus can be covered by just one rectangle. Therefore the cover size is $\Theta(an/a + 4an/a)$. For $a = \sqrt{n}$, the cover sizes are $\Theta(n\sqrt{n})$ and $\Theta(n)$, respectively.

Nonrectilinear polygons. When the polygon itself is not rectilinear but has only obtuse angles, suitably discretizing the problem so as to apply the greedy set covering algorithm [10] is itself nontrivial. Levcopoulos and Gudmundsson [14] showed that this can indeed be done. So this problem too has an $O(\log n)$ factor approximation algorithm, and no better bound is known.

Rectilinear polygons and fat rectangles. When the covering objects are squares or rectangles with bounded aspect ratio, then Levcopoulos and Gudmundsson [15] give a constant factor approximation algorithm.

Acknowledgment. We thank the referees for extensive comments that have helped to improve the presentation.

REFERENCES

[1] V.S. Anil Kumar, S. Arya, and R. Hariharan, *Hardness of set covering with intersection 1*, in Proceedings of the 27th International Colloquium on Automata, Languages and Programming, Geneva, Switzerland, 2000, pp. 624–635.

[2] P. Berman and B. Dasgupta, *Approximating rectilinear polygon cover problems*, Algorithmica, 17 (1997), pp. 331–356.

[3] H. Brönnimann and M. Goodrich, *Almost optimal set covers in finite VC-dimension*, Discrete Comput. Geom., 14 (1995), pp. 263–279.

[4] S. Chaiken, D.J. Kleitman, M. Saks, and J. Shearer, *Covering regions by rectangles*, SIAM J. Alg. Disc. Meth., 2 (1981), pp. 394–410.

[5] Y. Cheng, S.S. Iyengar, and R.L. Kashyap, *A new method for image compression using irreducible covers of maximal rectangles*, IEEE Trans. Software Engrg., 14 (1988), pp. 651–658.

[6] J.C. Culberson and R.A.Reckhow, *Covering polygons is hard*, J. Algorithms, 17 (1994), pp. 2–44.

[7] D.S. Franzblau, *Performance guarantees on a sweep-line heuristic for covering rectilinear polygons with rectangles*, SIAM J. Disc. Math., 2 (1989), pp. 307–321.

[8] D.S. Franzblau and D.J. Kleitman, *An algorithm for constructing regions with rectangles*, Inform. and Control, 63 (1984), pp. 164–189.

[9] A. Hegedüs, *Algorithms for covering polygons with rectangles*, Comput. Aided Geom. Design, 14 (1982), pp. 257–260.

[10] D.S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.

[11] J.M. Keil, *Minimally covering a horizontally convex orthogonal polygon*, in Proceedings of the 2nd Annual ACM Symposium on Computational Geometry, Yorktown Heights, NY, 1986, pp. 43–51.

[12] C. Levcopoulos, *A fast heuristic for covering polygons by rectangles*, in Proceedings of Fundamentals of Computer Theory, Lecture Notes in Comput. Sci. 199, Springer, Berlin, 1985, pp. 269–278.

[13] C. Levcopoulos, *Improved bounds for covering general polygons by rectangles*, in Proceedings of 6th Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 287, Springer, Berlin, 1987, pp. 95–102.

[14] C. Levcopoulos and J. Gudmundsson, *Close approximations of minimum rectangular covering*, in Proceedings of 16th Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 1180, Springer, Berlin, 1996, pp. 135–146.

[15] C. Levcopoulos and J. Gudmundsson, *Approximation algorithms for covering polygons with squares and similar problems*, in Proceedings of RANDOM'97, Lecture Notes in Comput. Sci. 1269, Springer, Berlin, 1997, pp. 27–41.

[16] L. Lovasz, *On the ratio of optimal integral and fractional covers*, Discrete Math., 13 (1975), pp. 383–390.

[17] R. Raz and S. Safra, *A sub-constant error-probability low-degree test and a sub-constant error-probability PCP characterization of NP*, in Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997.

[18] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 286–293.

[19] W.J. Masek, *Some NP-Complete Set Covering Problems*, manuscript, MIT, Cambridge, MA, 1979.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1542–1569

ON THE AUTOREDUCIBILITY OF RANDOM SEQUENCES*

TODD EBERT†, WOLFGANG MERKLE‡, AND HERIBERT VOLLMER§

Abstract. A binary sequence $A = A(0)A(1)\ldots$ is called infinitely often (i.o.) Turing-autoreducible if A is reducible to itself via an oracle Turing machine that never queries its oracle at the current input, outputs either $A(x)$ or a don't-know symbol on any given input x, and outputs $A(x)$ for infinitely many x. If in addition the oracle Turing machine terminates on all inputs and oracles, A is called i.o. truth-table-autoreducible.

We obtain the somewhat counterintuitive result that every Martin-Löf random sequence, in fact even every rec-random or p-random sequence, is i.o. truth-table-autoreducible. Furthermore, we investigate the question of how dense the set of guessed bits can be when i.o. autoreducing a random sequence. We show that rec-random sequences are never i.o. truth-table-autoreducible such that the set of guessed bits has positive constant density in the limit and that a similar assertion holds for Martin-Löf random sequences and i.o. Turing autoreducibility. On the other hand, we show that for any rational-valued computable function r that goes nonascendingly to zero, any rec-random sequence is i.o. truth-table-autoreducible such that on any prefix of length m at least a fraction of $r(m)$ of the m bits in the prefix are guessed.

We include a self-contained account of the hat problem, a puzzle that has received some attention outside of theoretical computer science. The hat problem asks for guessing bits of a finite sequence, thus illustrating the notion of i.o. autoreducibility in a finite setting. The solution to the hat problem is then used as a module in the proofs of the positive results on i.o. autoreducibility.

Key words. random sequences, autoreducibility, infinitely often autoreducibility, density of guessed bits, hat problem, Martin-Löf random sequences, rec-random sequences, p-random sequences, Turing autoreducibility, truth-table autoreducibility, error-correcting codes, hypercube topology

AMS subject classifications. 03D15, 03D30, 68P30, 68Q30, 91A60

DOI. 10.1137/S0097539702415317

1. Introduction.

In probability theory one fundamental idea is the concept of independence. A collection of random variables $X_1, X_2, \ldots$ is independent if the information obtained from observing the outcome of the variables X_j where $j \neq i$ leaves the distribution of X_i unaffected, in that the a posteriori distribution of X_i equals its a priori distribution. Moreover, viewed from a computational standpoint this idea can be translated as saying that an algorithm whose goal on input i is to guess or estimate the outcome of X_i should not benefit from querying about the outcome of the X_j where $j \neq i$. For example, consider the chance experiment where the bits of an infinite binary sequence $R(0)R(1)R(2)\ldots$ are obtained by successive tosses of a fair coin. If we want to come up with a procedure that on input i computes the bit $R(i)$ while having access only to the remaining bits of R but not to $R(i)$, the a posteriori probability of guessing $R(i)$ given knowledge of $R(j)$ for all $j \neq i$ equals the a priori probability of guessing $R(i)$; thus the chance of success when guessing $R(i)$ cannot be better than $1/2$, and the probability that all bits of R are guessed correctly by the given rule is 0. In particular, it seems natural to regard the information obtained

*Received by the editors September 19, 2002; accepted for publication (in revised form) July 2, 2003; published electronically October 2, 2003.
http://www.siam.org/journals/sicomp/32-6/41531.html
†Department of Computer Engineering and Computer Science, California State University, Long Beach, Bellflower Blvd., Long Beach, CA 90840 (ebert@cecs.csulb.edu).
‡Ruprecht-Karls-Universität Heidelberg, Institut für Informatik, Im Neuenheimer Feld 294, 69120 Heidelberg, Germany (merkle@math.uni-heidelberg.de).
§Universität Hannover, Theoretische Informatik, Appelstraße 4, 30167 Hannover, Germany (vollmer@informatik.uni-hannover.de).

from observing $R(j)$ with $j \neq i$ as unhelpful for guessing $R(i)$. However, somewhat counterintuitively, we demonstrate in this paper that there are algorithms that with probability 1 are, infinitely often and without error, capable of guessing the outcome of $R(i)$ by querying an oracle about the outcomes of $R(j)$, $i \neq j$.

For the moment, say an effective procedure with limited access to R as described above *autoreduces* R in case the procedure computes $R(i)$ for all i, and the procedure *infinitely often (i.o.) autoreduces* R in case the procedure computes $R(i)$ for infinitely many i, while for all other inputs the procedure eventually signals ignorance about the correct value. Then it appears that for any effective procedure it is impossible to autoreduce or even to i.o. autoreduce R because at first glance it would seem certain that in the limit, half of the membership guesses must be wrong. Indeed, by Corollary 6.5 below, for almost all sequences R (i.e., with probability 1) the sequence R cannot be autoreduced. However, and this comes as a slight surprise, almost all sequences R can be i.o. autoreduced according to Theorem 5.1.

But how can it be that almost all sequences can be i.o. autoreduced when the bits of these sequences, hence in particular all the bits guessed, are chosen independently of all the other bits? Recall that by the strong law of large numbers, with probability 1 the frequency $(R(0)+\cdots+R(n-1))/n$ of 1's in R converges to $1/2$. Furthermore, given a sequence of subsets of the natural numbers where the kth set has cardinality k, the Borel–Cantelli lemma tells us that with probability 1, at most finitely many of these sets have an empty intersection with R [38]. This shows that independent random events considered collectively may possess certain properties with probability 1. Thus we may assume that certain properties are present in almost all sequences, and for appropriate properties this can be exploited in order to compute certain bits of a sequence. In summary, the crux of the following investigation rests on determining degrees to which properties that are present in almost all sequences may be used to occasionally compute the outcome of a random variable by observing the outcomes of other random variables where the random variables are mutually independent.

Being almost convinced that random sequences might indeed be i.o. autoreducible, we might still wonder how we can overcome the obstacle that when guessing $R(i)$ for given places i, necessarily we err half the time. The key observation is that a procedure that i.o. autoreduces R can decide on its own whether to make a guess on a certain input i. Then, by assuming an appropriate property that is present in almost all sequences, for all such sequences an effective procedure may compute infinitely many bits, despite its querying limitations. Observe in this connection that for the autoreductions to be constructed in the sequel, for almost all sequences the places that are guessed form a set that is not computable. Furthermore, the set of guessed bits cannot have constant positive density in the set of all words.

In what follows, we use the known concept of autoreducibility by oracle Turing machines for capturing the idea of using a sequence A as oracle in order to compute A, yet not being able to query A about the bit to be computed. The concept of i.o. autoreducibility, where just infinitely many bits of A have to be computed, is modeled by means of oracle Turing machines that, in addition to 0 and 1, may also output a special don't-know symbol. Moreover, we will consider restricted concepts of autoreducibility and i.o. autoreducibility that correspond to various reducibilities considered in recursion theory and complexity theory. For example, we introduce i.o. tt-autoreducibility, which is defined similarly to the usual truth-table reducibility from recursion theory.

Furthermore, we do not just show that almost all sequences can be i.o. autoreduced by effective reductions of truth-table type, but, and this is more, the latter as-

sertion holds for all Martin-Löf random, rec-random, and even p-random sequences. The positive results on general i.o. autoreducibility are complemented by negative results on i.o. autoreducibility, where, for example, a positive constant fraction of all bits has to be guessed correctly. These negative results exhibit interesting interactions between the type of random sequence considered and the type of autoreduction employed, intuitively speaking.

The aim and scope of this paper can then be summarized as follows. We try to contribute to the investigation of the question of which types of random sequences are i.o. autoreducible, at which density, and with respect to which types of reducibilities.

We conclude this section with an outline of the paper and an overview on its technical contributions. In section 2 we state a puzzle, the *hat problem*, also known as *colored hat problem* or *prisoners' problem*. By means of the hat problem we illustrate how techniques from coding theory can be applied when trying to autoreduce random sequences. More precisely, we review perfect one-error-correcting codes and show that these codes can be used to derive optimal solutions for certain instances of the hat problem. In subsequent sections, the solutions are then used as basic modules when constructing autoreductions. The hat problem was introduced by Ebert [19] in order to illustrate the problem of autoreducing random sequences and has recently become well known outside of theoretical computer science [14, 35, 37]. In an attempt to provide a reference for the hat problem that is also accessible to readers that are not interested in applications to autoreducibility, we have tried to make section 2 self-contained.

In section 3 we review effective random sequences and related issues in effective measure theory, and in section 4 we give formal definitions for the concepts of autoreducibility that are subsequently used.

In section 5, we consider i.o. autoreducibility of random sequences. We prove that *every* rec-random sequence is i.o. truth-table-autoreducible and, what is more, in fact any p-random sequence is i.o. truth-table-autoreducible via an oracle Turing machine that runs in polynomial time. As mentioned above, this result seems somewhat surprising and even paradoxical in that the machine that witnesses the autoreducibility has the task of infinitely often guessing a bit of a random sequence and guessing correctly each time despite a high chance of error for each guess. We then show that these results require Turing machines where the number of queries made for a single input is unbounded. This is accomplished by proving that no rec-random sequence is i.o. bounded truth-table-autoreducible, i.e., i.o. autoreducible by an oracle Turing machine that is restricted to some fixed number of queries, and an analogous result is shown in a setting of polynomial time bounds.

In section 6, we introduce the notion of autoreducibility with density $r(m)$ as a gauge of how often an oracle machine can guess the bits of a random sequence or, in other words, how dense the set of guessed bits can be with respect to the entire set of bits. A sequence is i.o. autoreducible with density $r(m)$ if it is i.o. autoreducible such that for all m, at least a fraction of $r(m)$ of the first m bits of the sequence is guessed. In Theorem 6.3 it is shown that rec-random sequences are never i.o. truth-table-autoreducible with positive constant density (i.e., with density $r(m) = \varepsilon m$ for some $\varepsilon > 0$) and that a similar assertion holds with respect to Martin-Löf random sequences and i.o. Turing autoreducibility. On the other hand, Theorem 6.6 asserts that for any computable function r that goes nonascendingly to 0, any rec-random sequence is i.o. truth-table-autoreducible with density $r(m)$. So we obtain essentially matching bounds on the density of guessed bits for i.o. truth-table autoreductions of

rec-random sequences.

Related work. The current article is an extended joint version of conference articles by Ebert and Vollmer [21] and Ebert and Merkle [20]. The hat problem was originally introduced in the literature by Ebert [19] in order to illustrate the problem of autoreducing random sequences. The hat problem has led to work in coding theory [26] since there are instances of the problem for which the optimal solution (code) is not known. Moreover, the hat problem has become well known outside of theoretical computer science [14, 35, 37]. Independently and considerably earlier, towards the end of the 1980s, Aspnes et al. [9] considered voting problems that have a flavor similar to the hat problem. Moreover, Rudich [36] points out that the hat problem is essentially the same as a variant of the voting problems called "voting with abstention," and he reports unpublished earlier work on the latter. The concept of i.o. autoreducibility has been investigated by Beigel, Fortnow, and Stephan [10], who construct a sequence in exponential time that is not i.o. truth-table-autoreducible in polynomial time.

Notation. We use standard notation, which is elaborated further in the references [6, 11, 12, 29].

We consider words over the binary alphabet $\{0,1\}$, which are ordered by the usual length-lexicographical ordering; the $(i+1)$st word in this ordering is denoted by s_i; hence, for example, s_0 is the empty word λ. Occasionally, we identify words with natural numbers via the mapping $i \mapsto \mathrm{s}_i$.

If not explicitly stated differently, a sequence is an infinite binary sequence, and a class is a set of sequences. A subset A of the natural numbers $\mathbb{N}$ is identified with its characteristic sequence $A(0)A(1)\ldots$, where $A(x)$ is 1 if $x \in A$ and $A(x)$ is 0 otherwise; notation defined for such subsets is extended to the corresponding sequences; e.g., an oracle Turing machine may reduce one sequence to another. The term class refers to a set of sequences.

An *assignment* is a (total) function from some subset of the natural numbers to $\{0,1\}$. An assignment is *finite* if and only if its domain is finite. An assignment with domain $\{0,\ldots,n-1\}$ is identified in the natural way with a word of length n. For an assignment σ with domain $\{z_0 < \cdots < z_{n-1}\}$, the *word associated with* σ is the (unique) word w of length n that satisfies $w(i) = \sigma(z_i)$ for $i = 0,\ldots,n-1$.

The restriction of an assignment σ to a set I is denoted by $\sigma|I$. In particular, for any sequence X, the assignment $X|I$ has domain I and agrees there with X. For a sequence X and an assignment σ, we write $\langle X,\sigma\rangle$ for the sequence that agrees with σ for all arguments in the domain of σ and agrees with X otherwise.

The class of all sequences is referred to as *Cantor space* and is denoted by $\{0,1\}^\infty$. The class of all sequences that have a word x as a common prefix is called the *cylinder generated by* x and is denoted by $x\{0,1\}^\infty$. For a set W, let $W\{0,1\}^\infty$ be the union of all the cylinders $x\{0,1\}^\infty$ where the word x is in W.

We write Prob[.] for probability measures and $\mathbf{E}$[.] for expected values. Unless stated otherwise, all probabilities refer to the *uniform measure* (or *Lebesgue measure*) on Cantor space, which is the probability distribution obtained by choosing the individual bits of a sequence by independent tosses of a fair coin. Usually we write Prob[A satisfies ...] instead of Prob[$\{A \in \{0,1\}^\infty : A$ satisfies ...$\}$] in case it is understood from the context that the measure is with respect to A.

Logarithms are to base 2. The function $\langle .,. \rangle$ from $\mathbb{N}\times\mathbb{N}$ to $\mathbb{N}$ is the usual effective and effectively invertible pairing function [42].

2. The hat problem and error-correcting codes. This section features the hat problem, which asks for guessing a single bit of a randomly chosen finite sequence from the remaining bits, a problem that resembles the task of i.o. autoreducing random infinite sequences. Subsequently, in constructions that solve the latter task, the hat problem and its solution are used as a basic module. The hat problem is formulated as a puzzle, and as such has received some attention in the public [14, 35, 37].

The hat problem.

In the hat problem for a team of n players, a binary sequence of n bits is chosen by independent tosses of a fair coin. Player i is assigned the ith bit (or, equivalently, is assigned one of two possible hat colors according to this bit).

Afterwards, each player may cast a guess on its own bit (or may abstain from guessing) under the following conditions. The players know the bits of all other players but not their own bit. Once the game begins, the players are neither allowed to communicate nor do they know whether the other players have already guessed. However, the players can meet for a strategy session before the game begins.

The team wins if and only if there is at least one correct and no incorrect guess.

At first sight, since each player may observe only events that are independent of his own hat color, one might expect that the team should have no more than a 50% chance of winning. However, since they converse before the game, we demonstrate how collaboration can increase their chances.

EXAMPLE 2.1. *Consider the hat problem with $n = 3$ players, and suppose the team agrees on the following guessing strategy. Upon the start of the game, each player observes the hats of his two teammates. If both hats have the same color, then the player guesses his hat is colored differently. However, if the hats have different colors, then he passes.*

To compute the chances for a win under this strategy, we distinguish two cases. The first case occurs when all three hats have the same color, i.e., for two out of the 8 equiprobable assignments. In this case, each player will incorrectly guess. The second case is exactly two of the three hats are colored the same. Here exactly one player will venture a guess; this guess is correct, and the team wins. Hence the probability of winning using the above strategy is $1 - 2/8 = 3/4$.

Next we want to extend the solution to the hat problem with three players given in Example 2.1 to other team sizes. For a given team size n, identify assignments of colors with words of length n in the natural way; i.e., the players are numbered from 1 to n, and the jth bit of the word represents player j's hat color. The word that represents the actual assignment of colors is called the *true word*. With the true word understood, there are exactly two words of length n that agree with player j's view: the true word and the word that differs from the true word exactly at the jth position. Call these two words the *consistent words* of player j.

Specifying a strategy amounts to determining for any player and for any possible pair of consistent words for this player whether the player should cast a vote and, if yes, in favor of which of the two consistent words. This means that a strategy can be pictured as a directed graph G in the following way. The nodes of the graph are just the possible assignments, i.e., the words in $\{0,1\}^n$. The graph contains an edge from u to v if and only if these two nodes may occur as the consistent nodes of some player,

and in this situation the player votes in favor of v. Formally, we have $G = (V, E)$ where

$$V = \{0,1\}^n, \qquad E \subseteq \{(u,v) \mid u \text{ and } v \text{ differ exactly in one position }\},$$

and for any pair u and v of nodes in V, at most one of the edges (u,v) and (v,u) is in E.

Consider any strategy and its associated graph, and assume that the strategy is applied in a situation where the true word is u. The team wins if according to the given strategy some player casts a vote in favor of u but no player guesses in favor of a word different from u. In terms of the associated graph, this means that the team wins on the assignment u if and only if

(2.1) some edge is pointing to u and no edge is pointing away from u.

From this characterization of winning assignments we obtain an equivalent formulation of the hat problem as a network problem, which is stated in Remark 2.2. Afterwards, we construct solutions to this network problem and translate them back to the hat problem. The point in considering the network problem is that the way its solutions work is more easily understood than for the hat problem.

REMARK 2.2. *The hat problem can be reformulated as a problem on communication networks with a hypercube topology, where the nodes of the network correspond to the possible assignments of hat colors to the players.*

In order to obtain an equivalent version of the hat problem with n players, we consider a network of 2^n processors or nodes. The nodes are connected according to a hypercube topology [25]. *That is, each node is labelled by a unique word of length n, and between any two distinct nodes there is a link if and only if their labels differ in exactly one bit. The links are capable of transmitting information in either direction but only in one direction at a time. The task is to give a pattern of communication such that there is a maximum number of nodes u such that*

(2.2) *some node is sending to u and u is not sending to any node.*

A pattern of communication specifies for each link either the status idle or an orientation, i.e., one of the two possible directions of sending.

Any pattern of communication translates naturally into a strategy for the hat problem with n players and vice versa. Furthermore, under this translation the fraction of nodes that satisfy (2.2) *coincides with the success probability of the strategy. For a proof of the two latter assertions it suffices to observe that the representation of a strategy as directed graph is essentially identical to a pattern of communication and to compare the conditions* (2.1) *and* (2.2).

EXAMPLE 2.3. *The solution of the hat problem with $n = 3$ players from Example* 2.1 *translates as follows into a solution of the network problem. The network has a hypercube topology with 2^3 nodes, and the pattern of communication specifies that*

> *exactly the nodes* 000 *and* 111 *are sending, and each of them sends to all its neighbors in the hypercube.*

This way every processor u with a label different from 000 *and* 111 *is receiving but does not send and thus satisfies* (2.2).

In order to extend the solution to the network problem given in Example 2.3 to larger networks, we review some notation and facts from coding theory. The

(Hamming) distance $\mathrm{d}(u,v)$ between two words u and v of identical length is the number of positions at which u and v differ. Furthermore, the *unit ball* with *center* u is the set of all words that have the same length as u and differ from u at most at one position, i.e., the set

$$\{v : \mathrm{d}(u,v) \leq 1\}.$$

The *surface* of a unit ball is just the ball with its center removed.

In the network problem, the set of neighbors of a node w coincides with the surface of the unit ball centered at w. Accordingly, the solution to the network problem from Example 2.3 can be reformulated as follows.

> Exactly the nodes 000 and 111 are sending, and each of them sends to all nodes on the surface of the unit ball centered at this node.

The easy idea underlying this solution works also for networks where the parameter n is larger than 3. Just select a subset C of all words of length n, and let each node (labelled by a word) in C send to all nodes that are on the surface of the unit ball centered at this node but are not in C themselves. If, for example, we choose the set C such that the unit balls around the words in C are disjoint, then exactly the nodes on the surfaces of these unit balls satisfy (2.2); i.e., these are the nodes that receive but do not send. The fraction of such nodes becomes maximum among all corresponding choices of C in case the unit balls around the words in C partition the set of all words of the given length. Such partitions are studied in coding theory under the name of perfect one-error-correcting codes.

DEFINITION 2.4. *Any subset of* $\{0,1\}^n$ *is called a* code *with* (codeword) length n. *A code* C *with length* n *is called* perfect one-error-correcting *if the unit balls around the codewords in* C *form a partition of* $\{0,1\}^n$ *(i.e., if for any word* w *of length* n *there is exactly one word* $c \in C$ *such that* $\mathrm{d}(w,c) \leq 1$*).*

For any perfect one-error-correcting code of length n, the number of all words 2^n must be divisible by the unit ball volume $n+1$; hence $n+1$ must be a power of 2. It is well known that this necessary condition on the codeword length is also sufficient [22, 45].

FACT 2.5. *Let* n *be of the form* $2^k - 1$ *where* $k > 0$ *is a natural number. Then there is a perfect one-error-correcting code* C_n *of codeword length* n. *Furthermore, the codes* C_n *can be chosen such that their union* $\bigcup_{\{n:n=2^k-1\}} C_n$ *is decidable in polynomial time. For example, such codes are given by the well-known family of binary Hamming codes* [45].

Proof. We identify words and binary vectors in the obvious way; hence the words of any given length form a vector space under addition modulo 2 and over the field with two elements. Let M be the $k \times n$ binary matrix where for $i = 1, \ldots, n$, column i of M is the binary representation of the number i. The matrix M defines a linear mapping $w \mapsto Mw$ from words of length n to words of length k. Let C_n denote the kernel space of this mapping, i.e., the set of all words that are mapped to 0^k. Then C_n is a perfect one-error-correcting code of length n.

For a proof, first observe that trivially every word in the kernel has length n. Second, any word w of length n is contained in a unit ball centered at a word in C_n. In case Mw is zero, this is obvious. Otherwise, Mw appears as a column of M, say, as column j, hence flipping the jth bit of w results in a word in C_n. Third, the unit balls centered at the words in C_n are mutually disjoint. Fix any two distinct words u and v in C_n. Then M maps both u and v to 0^k, and, by linearity of matrix multiplication,

the same holds for $u-v$. On the other hand, $M(u-v)$ is just the sum over all column vectors j of M such that u and v differ at the jth position. Now the sum over one or over two distinct column vectors of M cannot be equal to 0^k; hence $\mathrm{d}(u,v) \geq 3$, and, in particular, the unit balls centered at u and v must be disjoint. (The third item is a special form of a well-known result in coding theory that the minimum distance of a code which is the kernel space of some matrix M is equal to d, where d is the least number for which there are d linearly dependent column vectors of M. In our case it is easy to check that M has three linearly dependent vectors, but not two. Thus C_n has a minimum distance of 3.)

Finally, the problem of deciding if a word belongs to $\bigcup_{\{n:n=2^k-1\}} C_n$ can be solved in polynomial time, as essentially it involves only multiplying the input with a matrix that has moderate size and is easy to compute. □

Remark 2.6 summarizes the application of error-correcting codes to the hat problem and the related network problem.

REMARK 2.6. *Let n be of the form 2^k-1 for some natural number $k>0$, and consider the hat problem and the network problem with parameter n. Fix a perfect one-error-correcting code of code word length n, and call the words in this code, as well as the nodes labelled by these words,* designated.

A solution to the network problem is given by the following pattern of communication. By Remark 2.2*, under this pattern of communication exactly the nodes that are* not *designated satisfy* (2.2).

> *Every designated node sends to all its neighbors in the hypercube; the other nodes do not send.*
> *(That is, every designated node w sends to all nodes on the surface of the unit ball centered at w.)*

A solution to the hat problem with n players is given by the strategy where each player behaves according to the following rule. By Remark 2.2*, under this strategy the team wins exactly for the assignments that are* not *designated.*

> *In case one of the consistent words is a designated word, venture a guess according to the assumption that the true word is the other consistent word; otherwise, pass.*
> *(That is, in case the consistent words are a designated word w and a word on the surface of the ball centered at w, guess in favor of the consistent word on the surface, i.e., the one different from w.)*

The fraction of designated words is $1/(n+1)$ because the balls centered at the designated words partition $\{0,1\}^n$, and each such ball consists of one designated and n other words. Hence if this strategy is applied in the hat problem with n players, the team wins with probability $1-1/(n+1)$.

EXAMPLE 2.7. *Consider the hat problem with $n=7$ players. For the strategy described in Remark* 2.6*, the codewords comprise a fraction of* $1/8$ *of the* 128 *words in $\{0,1\}^7$; hence there are* 16 *codewords and* 112 *error words, and the team wins with probability* $112/128 = 0.875$.

In the network problem with parameter n, a node can send at most to n other nodes; hence, among all nodes that send or receive, at least a fraction of $1/(n+1)$ nodes must send. The pattern of communication described in Remark 2.6 achieves this bound and thus is an optimum solution to the network problem; hence by the discussion in Remark 2.2 also the corresponding strategy for the hat problem is optimum.

Remark 2.8 contains an alternate, somewhat more formal proof for the optimality of this strategy, which features the idea that for any strategy the expected number of correct and incorrect guesses is the same.

REMARK 2.8. *Let n be of the form $n = 2^k - 1$. For the hat problem with n players, the probability of success of $1 - 1/(n+1)$ that is achieved by the strategy described in Remark* 2.6 *is optimum.*

For a proof, fix any strategy. Recall that the colors of the hats are assigned according to independent tosses of a fair coin; hence, whenever the strategy tells a player to guess, the probability of a correct guess is exactly $1/2$. *As a consequence, the expected number of correct and incorrect guesses per player is the same, and, by linearity of expectation, the same holds for the entire team; i.e., if we define the random variables g_c and g_i as equal to the number of correct and incorrect guesses, respectively, for the entire team, their expected values $\mathbf{E}[g_c]$ and $\mathbf{E}[g_i]$ coincide. Furthermore, if the strategy considered has probability of success of p, then we have*

$$p \leq \mathbf{E}[g_c] = \mathbf{E}[g_i] \leq (1-p)n. \tag{2.3}$$

In (2.3)*, the equation holds by the preceding discussion, the left-hand inequality follows because for each assignment that leads to a win there must be at least one correct guess, and the right-hand inequality holds because for any winning assignment there is no wrong guess, while for any other assignments there are at most n wrong guesses. So we have $p \leq (1-p)n$, and by rearranging we obtain $p \leq 1 - 1/(n+1)$.*

Lenstra and Seroussi [26] discuss applications of coding theory to the hat problem. They investigate good strategies for numbers of players that are not of the form $2^k - 1$ and for more general versions of the hat problem with more than two colors. For the hat problem with two colors, they show that strategies are equivalent to covering codes. Their observation is reviewed in Remark 2.9, where we give an equivalent formulation in terms of communication patterns for the network problem.

REMARK 2.9. *A code of length n is called a covering code (more precisely, a* 1*-covering code) if any word of length n differs at most at one position from some word in the code. For example, perfect one-error-correcting codes are covering codes; however, in general the unit balls centered at the words in a covering code are not mutually disjoint.*

Given a pattern of communication for the network problem with parameter n, let C be the set of all nodes that do not satisfy (2.2)*; i.e., C contains the nodes that are sending or are neither sending nor receiving, and the complement of C contains the nodes that receive but do not send. Then C is a covering code because the nodes not in C are receiving; hence each such node must be at distance at most* 1 *from a sending node, which then must be a node in C. Conversely, given any covering code of length n, there is a pattern of communication where every node in C sends to all its neighbors that are not in C themselves. With this pattern, exactly the nodes that are not in C satisfy* (2.2)*.*

3. Random sequences. This section gives a brief introduction to the theory of effective measure. We focus on effective random sequences and related concepts that are used in the following. For more comprehensive accounts of effective measure we refer the reader to the references [5, 6, 29].

Imagine a casino that offers roulette, and consider the sequence of outcomes red and black that occur in the course of the game. We would certainly not call this sequence random if there were a method to determine any next bit before the corresponding drawing has actually taken place. But also if we just knew a strategy

that guarantees winning an unbounded amount of money when starting with finite initial capital, this would indicate that the sequence is nonrandom. So we might be tempted to call a sequence nonrandom if there is such a strategy. The problem with this definition is that for any sequence there is a strategy that wins against this sequence, e.g., the one that works by always predicting correctly the next bit of the sequence. However, the latter is not a problem for real casinos because for them a sequence is "random enough" if it does not permit a winning strategy that a gambler can actually play. In general, this suggests defining randomness relative to a certain class of admissible betting strategies instead of striving for an absolute concept. In what follows, the admissible betting strategies are just the ones that are computable in a specific model of computation. A sequence is called random with respect to such a model of computation if none of the admissible betting strategies leads to an unbounded gain when playing against this sequence.

In order to formalize the ideas of the preceding paragraph, consider the following gamble. Imagine a player that successively places bets on the individual bits of an unknown sequence A. The betting proceeds in rounds $i = 1, 2, \ldots$. During round i, the player receives as input the length $i-1$ prefix of A and then, first, decides whether to bet on the ith bit being 0 or 1 and, second, determines the stake that shall be bet. The stake might be any fraction between 0 and 1 of the capital accumulated so far; i.e., in particular, the player is not allowed to incur debts. Formally, a player can be identified with a *betting strategy*

$$b\colon \{0,1\}^* \to [-1,1]$$

where on input w the absolute value of $b(w)$ is the fraction of the current capital that shall be at stake, and the bet is placed on the next bit being 0 or 1 depending on whether $b(w)$ is negative or nonnegative.

The player starts with positive, finite capital. At the end of each round, in case of a correct guess, the capital is increased by that round's stake and, otherwise, is decreased by the same amount. So given a betting strategy b, we can inductively compute the corresponding *payoff function* d_b by applying the equations

$$d_b(w0) = d_b(w) - b(w)\cdot d_b(w), \qquad d_b(w1) = d_b(w) + b(w)\cdot d_b(w).$$

Intuitively speaking, the payoff $d_b(w)$ is the capital the player accumulates until the end of round $|w|$ by betting on a sequence that has the word w as a prefix. The payoff function d_b satisfies the fairness condition

$$d_b(w) \;=\; \frac{d_b(w0) + d_b(w1)}{2}\;. \tag{3.1}$$

We call a function d from words to nonnegative reals a *martingale* if and only if $d(\lambda) > 0$ and d satisfies the fairness condition (3.1), with d_b replaced by d, for all words w. By the discussion above, for a betting strategy b the function d_b is always a martingale, and, conversely, it can be shown that every martingale has the form d_b for some betting strategy b. Hence betting strategies and martingales are essentially equivalent. Accordingly, we extend occasionally notation defined for betting strategies to martingales and vice versa.

DEFINITION 3.1. *A betting strategy b* succeeds *on a sequence A if the corresponding martingale d_b is unbounded on the prefixes of A, i.e., if*

$$\limsup_{n\in\mathbb{N}}\; d_b(A|\{0,\ldots,n\}) \;=\; \infty.$$

In what follows, we will consider computable betting strategies. Any computable betting strategy b is confined to rational values, and there is a Turing machine that on input w outputs some appropriate finite representation of $b(w)$.

Computable betting strategies are not only of interest in connection with the definition of random sequences but are also the basis of the theory of effective measure. A betting strategy is said to *succeed on* or to *cover* a class if and only if it succeeds on every sequence in the class. Ville demonstrated that a class has uniform measure 0 if and only if the class can be covered by some, not necessarily effective, betting strategy [6, 47]. This result justifies the following notation. A class has *measure* 0 with respect to a given class of betting strategies if and only if it is covered by some betting strategy in the class. By appropriately restricting the class of admissible betting strategies, one obtains restricted concepts of measure 0 classes, which are useful when investigating classes occurring in recursion theory or complexity theory. Most of these classes are countable and hence have uniform measure 0; i.e., from the point of view of uniform measure all these classes have the same size. However, given a specific class $\boldsymbol{C}$, we might try to restrict the class of admissible betting strategies such that the resulting concept of measure 0 class is interesting in the sense that we can still cover relevant subclasses of $\boldsymbol{C}$ but not the class $\boldsymbol{C}$ itself. In the context of recursion theory, this led to the consideration of computable betting strategies [5, 39, 40, 43, 48]. In connection with complexity classes one imposes additional resource bounds [6, 28, 29, 31]; e.g., in the case of the class $\mathbf{E}$ of sequences that can be computed in deterministic linear exponential time, i.e., in time $2^{\mathrm{O}(n)}$, Lutz proposed using betting strategies that are computable in polynomial time.

REMARK 3.2. *The resources needed to compute a betting strategy are measured with respect to the length of the input w; for example, a betting strategy b is computable in polynomial time if $b(w)$ can be computed in time $|w|^c$ for some constant c.*

A prefix w of a sequence A encodes $A(\mathrm{s}_0)$ through $A(\mathrm{s}_{|w|-1})$, and accordingly on input w, a betting strategy determines a bet on whether $x = \mathrm{s}_{|w|}$ is in the unknown sequence or not. Observe that the length of x is approximately $\log|w|$; thus, for example, a time bound $|w|^c$ translates to a time bound of the form $2^{\mathrm{O}(|x|)}$.

After this short digression to effective measure theory we return to the endeavor of defining concepts of random sequences via restricted classes of betting strategies.

DEFINITION 3.3. *A sequence is* rec-random *if no computable betting strategy succeeds on it. A sequence is* p-random *if no betting strategy that is computable in polynomial time succeeds on this sequence.*

Besides p-random and rec-random sequences, we will consider Martin-Löf random sequences [30]. Let $W_0, W_1, \ldots$ be the standard enumeration of the computably enumerable sets [42].

DEFINITION 3.4. *A class $\mathcal{N}$ is called a* Martin-Löf null class *if and only if there exists a computable function $g\colon \mathbb{N} \to \mathbb{N}$ such that for all i*

$$\mathcal{N} \subseteq W_{g(i)}\{0,1\}^\infty \quad \textit{and} \quad \mathrm{Prob}[W_{g(i)}\{0,1\}^\infty] < \frac{1}{2^i}\,.$$

For such a function g, the sequence $W_{g(0)}, W_{g(1)}, \ldots$ is called a Martin-Löf null cover *for $\mathcal{N}$. A sequence is* Martin-Löf random *if it is not contained in any Martin-Löf null class.*

Martin-Löf random sequences have been characterized in terms of martingales by Schnorr [40]. A sequence is *Martin-Löf random* if and only if it cannot be covered by a subcomputable martingale. A martingale d is *subcomputable* if and only if there

is a computable function $\widetilde{d}$ in two arguments such that for all words w, the sequence $\widetilde{d}(w,0), \widetilde{d}(w,1), \ldots$ is nondecreasing and converges to $d(w)$.

REMARK 3.5. *For any sequence X we have*

$$X \text{ Martin-Löf random} \Rightarrow X \text{ rec-random} \Rightarrow X \text{ p-random}, \tag{3.2}$$

and both implications are strict.

The first implication in (3.2) *is immediate by the characterization of Martin-Löf random sequences in terms of subcomputable martingales and the observation that for a computable betting strategy the corresponding martingale is computable, too. Likewise, the second implication follows from the definitions of rec-random and p-random sequences in terms of computable and polynomial-time computable betting strategies. Furthermore, the second implication is strict because one can construct a computable p-random sequence by diagonalizing against an appropriate weighted sum of all betting strategies that are computable in polynomial time. The strictness of the first implication was implicitly shown by Schnorr* [40]. *For a proof, it suffices to recall that the prefixes of a Martin-Löf random sequence cannot be compressed by more than a constant* [27, *Theorem* 3.6.1] *while a corresponding statement for rec-random sequences is false* [27, 33].

By definition, a class $\mathcal{N}$ has uniform measure 0 if the condition in Definition 3.4 is satisfied for some arbitrary sequence of sets $V_0, V_1, \ldots$ in place of $W_{g(0)}, W_{g(1)}, \ldots$. Thus the concept of a Martin-Löf null class is indeed an effective variant of the classical concept of a class that has uniform measure 0. In particular, any Martin-Löf null class has uniform measure 0. By σ-additivity and since there are only countably many computable functions, the union of all Martin-Löf null classes has uniform measure 0. Accordingly, the class of Martin-Löf random sequences, and hence by Remark 3.5 also the classes of rec-random and of p-random sequences, have uniform measure 1. We note in passing that it can be shown that the union of all Martin-Löf null classes is again a Martin-Löf null class [17, section 6.2].

We conclude this section by describing a standard technique for the construction of betting strategies.

REMARK 3.6. *Let I be a finite set, and let Θ be a subset of all partial characteristic functions with domain I. Then there is a betting strategy that, by betting on places in I, increases its capital by a factor of $2^{|I|}/|\Theta|$ for all sequences B where the restriction of B to I is in Θ.*

The betting strategy is best described in terms of the corresponding martingale. The martingale takes the capital available when betting on the least element of I and distributes it evenly among the elements of Θ, and then computes values upwards according to the fairness condition for martingales.

4. Autoreducibility. For the moment, call a sequence X *autoreducible* if there is an effective procedure that on input x computes $X(x)$ while having access to the values $X(y)$ for $y \neq x$. Intuitively speaking, for an autoreducible sequence the information on $X(x)$ is not only stored at x but can also be effectively recovered from the remainder of the sequence. For example, any computable sequence is autoreducible, and for an arbitrary sequence Y, the sequence $Y(0)Y(0)Y(1)Y(1)\ldots$ is autoreducible. In a recursion theoretic setting, the concept of autoreducibility was introduced by Trakhtenbrot [44]. Further investigations showed, among other results, that autoreducibility is tightly connected to the concepts of mitoticity and introreducibility [8, 18, 23, 24]. The concept of autoreducibility was transferred to complexity theory by Ambos-Spies [2], and subsequently resource-bounded versions of autoreducibility

have been studied by several authors [10, 15, 16, 46]. Self-reducibility is a special form of autoreducibility where only queries less than the current input may be asked [11, 41]; it can be shown that certain forms of self-reducibility characterize certain types of generic sets [3, 4, 13].

Now consider the question of whether a random sequence can be autoreducible. By definition, the bits of an autoreducible sequence depend on each other in an effective way. This suggests that by exploiting the dependencies, we might come up with an effective betting strategy that succeeds on this sequence. Indeed Martin-Löf random sequences are never autoreducible, and, similarly, rec-random sequences are not autoreducible by reductions that are confined to nonadaptive queries; see Corollary 6.5 below.

Pushing the issue further, we might ask whether for a random sequence R it is at least possible to recover some of the values $R(x)$ from the values $R(y)$ with $y \neq x$. Trivially, this is possible for finitely many places x, so let us consider the case of infinitely many x. For the moment, call a sequence X *i.o. autoreducible* if there is an effective procedure that for infinitely many inputs x computes $X(x)$ while having access to the values $X(y)$ for $y \neq x$, whereas for all other inputs the procedure eventually signals that it cannot compute the correct value. For example, any sequence that, if viewed as a set, has an infinite computable subset is i.o. autoreducible; hence, in particular, any sequence that corresponds to an infinite computably enumerable set is i.o. autoreducible. Observe that by a standard diagonalization argument of finite extension type, one can easily construct a sequence that is computable in the halting problem and is not i.o. autoreducible.

For an i.o. autoreducible sequence R there are infinitely many places x where the value of $R(x)$ depends in an effective way on the remainder of the sequence R. On first sight, the situation looks rather similar to the case of an autoreducible sequence, and indeed it is tempting to assume that random sequences cannot be i.o. autoreducible. So the following result is somewhat surprising. *Every* p-random sequence is i.o. autoreducible by a reduction procedure that runs in polynomial time. This and related results are demonstrated in section 5. In the remainder of this section, we give formal definitions for various concepts of autoreducibility.

Recall the concept of an *oracle Turing machine* [11], which is a Turing machine that during its computation has access to a sequence X, the oracle. In case an oracle Turing machine M eventually terminates on input x and with oracle X, let $M(X, x)$ denote the computed value and, otherwise, i.e., if M does not terminate, say that $M(X, x)$ is undefined. But rather than use standard oracle machines M whose defined outputs $M(X, x)$ belong to $\{0, 1\}$, we also allow the machines to output a special "don't-know-symbol" $\perp$, which has the intended meaning of signaling that the correct value is not known.

DEFINITION 4.1. *Let M be an oracle Turing machine (with output in $\{0, 1, \perp\}$), and let A, B, and E be sequences. Then M* reduces *A to B on E if and only if*

(i) $M(B, x) = A(x)$ *for all x, where $M(B, x) \neq \perp$, and*
(ii) $M(B, x) \neq \perp$ *for all $x \in E$.*

If an oracle Turing machine M reduces a sequence A to a sequence B on an infinite set, we say that M *infinitely often reduces* or, for short, *i.o. reduces* A to B. If M reduces A to B on the set of all words, we say that M *reduces* A to B. Obviously, the latter notion coincides with the usual concept of reduction by a $\{0, 1\}$-valued oracle Turing machine, where it is required that $M(B, x)$ agrees with $A(x)$ for all x.

DEFINITION 4.2. *Let M be an oracle Turing machine, let A be a sequence, and let E be a set. The set of query words occurring during the computation of M on input x and with oracle A is denoted by* $\mathrm{Q}(M, A, x)$.

The sequence A is autoreduced on set E by M *if M reduces the sequence A to itself on E and $x \notin \mathrm{Q}(M, A, x)$ for all x. The sequence A is* i.o. autoreduced *by M if A is autoreduced by M on an infinite set. The sequence A is* autoreduced *by M if A is autoreduced by M on the set of all words.*

Next we define concepts of autoreducibility that correspond to the standard effective reducibilities considered in recursion theory [42] and to the standard reducibilities computable in polynomial time considered in complexity theory [11]. More precisely, for

$$r \in \{\mathrm{T}, \mathrm{tt}, \mathrm{btt}, \mathrm{btt}(k), \text{p-T}, \text{p-tt}, \text{p-btt}, \text{p-btt}(k)\},$$

we define the concepts of r-autoreducibility on a set E, of i.o. r-autoreducibility, and of r-autoreducibility.

For a start, we consider Turing- or, for short, T-autoreducibility. A sequence is *T-autoreducible on E* if it can be autoreduced on E by some oracle Turing machine M. A sequence is *i.o. T-autoreducible* if it is T-autoreducible on an infinite set, and a sequence is *T-autoreducible* if it is T-autoreducible on the set of all words.

The definitions for the remaining cases are basically the same; however, the oracle Turing machine M that performs the autoreduction has to, in addition, satisfy certain requirements. In particular, in the following M must always be total; i.e., on all inputs and for all oracles, M must eventually finish its computation. In the case of *truth-table autoreducibility* (tt), M has to ask its queries nonadaptively; i.e., M computes a list of queries that are asked simultaneously, and, after receiving the answers, M is not allowed to access the oracle again. In the case of *bounded truth-table autoreducibility* (btt), the queries have to be asked nonadaptively, while the number of queries that might be asked on a single input is bounded by a constant. Even more restrictive, in the case of *btt(k)-autoreducibility* the number of nonadaptive queries is bounded by the fixed constant k. The concepts of *polynomial time-bounded autoreducibility* like p-T- or p-tt-autoreducibility are defined accordingly where it is required in addition that M runs in polynomial time.

We conclude this section with some technical remarks on the representation of oracle Turing machines. By definition, tt-autoreductions are performed by total oracle Turing machines that query the oracle nonadaptively. Such an oracle Turing machine can be conveniently represented by a pair of computable functions g and h where $g(x)$ gives the set of words queried on input x and $h(x)$ specifies how the answers to the queries in the set $g(x)$ are evaluated; i.e., $h(x)$ tabulates a $\{0, 1, \perp\}$-valued function over $|g(x)|$ variables. In this situation we refer to $h(x)$ as a truth-table. Likewise, oracle Turing machines that witness a p-btt-autoreduction can be represented by pairs of functions g and h that are computable in polynomial time (whereas in general this is not possible for p-tt-autoreductions because the size of the corresponding truth-tables may be exponential in the input length).

REMARK 4.3. *Alternative to the $\{0, 1, \perp\}$-valued oracle Turing machine model, one could use the standard model, in that rather than output "don't-know," the Turing machine would simply query the oracle about the value of x and output that value. This formulation was originally used by Ebert* [19]*; a similar model is used by Arslanov* [8]*, with different notation and in the special case of a weak truth-table reduction that infinitely often queries the oracle only at places strictly less than the current input.*

We emphasize that the results of this paper hold for both the $\{0,1,\perp\}$-valued and the standard model; the main motivation for adopting the former is that it complies better with the usual classification of reducibilities according to whether they query the oracle adaptively or nonadaptively; i.e., the definitions of the various concepts of i.o. autoreducibility of truth-table type preserve the idea of accessing the oracle only once.

5. Autoreductions of random sequences. Next we apply the solution of the hat problem as discussed in section 2 to the construction of i.o. autoreductions of rec-random sequences.

THEOREM 5.1. *Every rec-random sequence is i.o. tt-autoreducible.*

Proof. Fix any rec-random sequence R. Partition the natural numbers into consecutive intervals $I_1, I_2, \ldots$ where I_k has size $l_k = 2^k - 1$. Write R in the form

$$R = w_1 w_2 \ldots \qquad \text{where for all } k, \quad |w_k| = l_k \ ;$$

i.e., w_k is the word associated with the restriction of R to I_k. Furthermore, for every $k > 0$ fix a perfect one-error-correcting code C_k of codeword length l_k such that given x, we can decide in polynomial time whether x is in one of the codes C_k.

In a nutshell, the proof of Theorem 5.1 works as follows. The code words in C_k comprise such a small fraction of all words of length l_k that in case infinitely many words w_k were in C_k there would be a computable betting strategy that succeeds on R. But R is assumed to be rec-random; hence w_k is not in C_k for almost all k. Then in order to construct an oracle Turing machine that witnesses that R is i.o. tt-autoreducible, we handle the intervals I_k individually and when working on I_k, we simulate the solution of the hat problem with l_k players. This way we can compute $R(x)$ for a single place x in I_k whenever w_k is not in C_k. But the latter is the case for almost all k; thus we are able to autoreduce R as required. Details follow.

CLAIM 1. *For almost all k, w_k is not in C_k.*

Proof. Consider the following betting strategy. For every $k > 0$, a portion $a_k = 1/2^k$ of the initial capital 1 is exclusively used for bets on words in the interval I_k. On each interval I_k, the betting strategy follows the strategy described in Remark 3.6, where C_k plays the role of Θ; i.e., the capital a_k is bet on the event that (the word associated with) the restriction of the unknown sequence to I_k is in C_k. By construction, just a fraction of $a_k = 1/(l_k + 1)$ of all words of length l_k belongs to C_k; hence the capital a_k increases to 1 for all k such that the restriction of the unknown sequence to I_k is in C_k. As a consequence, the betting strategy succeeds on any sequence such that for infinitely many k, the restriction of the sequence to the interval I_k is an element of C_k. But no computable betting strategy can succeed on the rec-random sequence R; hence Claim 1 follows. □

Observe that the proof of Claim 1 depends on the choice of the l_k only insofar as it is required that the sum over the a_k, where $a_k = 1/(l_k + 1)$, converges.

Next we define an oracle Turing machine M that witnesses that R is i.o. tt-autoreducible. By Claim 1, fix k_0 such that w_k is not in C_k for all $k > k_0$. On inputs in the intervals I_1 through I_{k_0}, M simply outputs $\perp$. On any input x in an interval I_k with $k > k_0$, M queries the oracle nonadaptively on all words in I_k that are different from x. Thereby M determines two words, u' and u'', and knows that w_k is equal to one of the words

$$v_0 = u'0u'' \quad \text{and} \quad v_1 = u'1u'' \ ,$$

where the uncertainty is with respect to the value of $R(x)$. Then M outputs i in case $v_{1-i} \in C_k$ and $v_i \notin C_k$, and, otherwise, i.e., if neither v_0 nor v_1 is in C_k, M outputs $\perp$.

By construction, M is computable, queries its oracle nonadaptively, and never queries its oracle at the input. On intervals I_k with $k \leq k_0$, M always outputs $\perp$. On any interval I_k with $k > k_0$, M simulates the strategy for the hat problem with l_k players from Remark 2.6 where C_k is the set of designated words. For any such interval, w_k is not in C_k by choice of k_0; hence the discussion in section 2 shows that M computes the correct value $R(x)$ at the single input x in the interval at which w_k differs from the closest code word in C_k, while M outputs $\perp$ for all other inputs in the interval. In summary, M i.o. tt-autoreduces R. □

Subsequently, the statement of Theorem 5.1 will be strengthened in various ways. As an immediate improvement, we note that the proof of Theorem 5.1 can be adjusted in order to obtain the following stronger but also more technical version of the theorem. Given a computable function t, we call a sequence *$t(m)$-random* if no betting strategy that is computable in time $\mathrm{O}(t(m))$ succeeds on this sequence, where m denotes the length of the prefix of the unknown set that a betting strategy receives as input.

THEOREM 5.2. *Let $q\colon \mathbb{N} \to \mathbb{N}$ be an unbounded and nondecreasing computable function. Every m^2-random sequence is i.o. tt-autoreducible by an oracle Turing machine M that on inputs of length n runs in time $\mathrm{O}(n)$ and asks at most $q(n)$ queries.*

Proof. The proof of Theorem 5.2 is rather similar to the proof of Theorem 5.1. We just indicate the necessary adjustments. Recall that the i.o. tt-autoreductions, as well as the betting strategy in the proof of Theorem 5.1 are built up from modules that work essentially independently on the intervals I_j. The key trick in the proof of Theorem 5.2 is to shift the intervals such that the length of the words contained in them are considerably larger than the length of the corresponding interval. More precisely, the interval I_k contains the first l_k words of length n_k, where the sequence $n_0, n_1, \ldots$ is chosen such that we have for all k,

$$\text{(i) } 2^{n_k} < n_{k+1}\ , \qquad \text{(ii) } 2^{l_k} < n_k\ , \qquad \text{(iii) } l_k < q(n_k).$$

In addition, we assume that there is a Turing machine that on input 1^n uses at most n steps to decide whether n appears in the sequence $n_0, n_1, \ldots$ and if so to compute the index k with $n = n_k$. Such a sequence can be obtained by standard methods as described in the chapter on uniform diagonalization and gap languages in Balcázar, Díaz, and Gabarró [11]. For example, we can first define a sufficiently fast growing time-constructible function $r : \mathbb{N} \to \mathbb{N}$ and then let n_i be the value of the i-fold iteration of r applied to 0.

Similar to Claim 1 in the proof of Theorem 5.1, we can argue that for any m-random sequence R and for almost all k, the restriction w_k of R to the interval I_k is not in the code C_k. Otherwise, there would be a betting strategy similar to the one used in the proof of the mentioned claim that wins on R; i.e., on any interval I_k the strategy bets a fraction of $1/2^k$ of its initial capital on the event that w_k is in C_n. By choice of the n_k, this strategy can be chosen to run in time m^2. Recall in connection with this time bound that the individual bets are specified as a fraction of the current capital; hence placing the bets related to interval I_k requires computing the outcomes of the bets on the previous intervals.

The sequence R is then i.o. tt-autoreducible by a reduction that simulates the solution to the hat problem on every interval I_k where w_k is not in C_k. On an input of length n, this reduction runs in time n and asks at most $q(n)$ queries because of (ii) and (iii), respectively. □

For a proof of the following corollary it suffices to observe that p-random sequences are in particular m^2-random, while every m^2-random sequence in turn is i.o. tt-autoreducible in polynomial time according to Theorem 5.2.

COROLLARY 5.3. *Every p-random sequence is i.o. p-tt-autoreducible.*

The next theorem shows that neither Theorems 5.1 and 5.2 nor Corollary 5.3 extends to i.o. btt-autoreducibility; i.e., the proofs of these results require oracle Turing machines that ask an unbounded number of queries.

THEOREM 5.4.

(a) *No rec-random sequence is i.o. btt-autoreducible.*

(b) *No p-random sequence is i.o. p-btt-autoreducible.*

Proof. (a) Fix any sequence A that is i.o. btt-autoreducible. It suffices to show that A is not rec-random, i.e., that there is a computable betting strategy b that succeeds on A.

Among all oracle Turing machines $M = (g, h)$ that i.o. btt-autoreduce A, we will consider only the ones that are *normalized* in the sense that the set of queries $g(x)$ is empty whenever the truth-table $h(x)$ is constant (i.e., whenever $h(x)$ evaluates to the same value for all assignments on $g(x)$). Furthermore, among all normalized oracle Turing machines that i.o. btt-autoreduce A, let $M = (g, h)$ be one such that its norm

$$l = \sup_{x \in \{0,1\}^*} |g(x)|$$

is minimum. In case $l = 0$, M is independent of the oracle, and we can compute $A(x)$ for the infinitely many places x where the reduction does not yield $\perp$; hence A is not rec-random, and we are done. So assume $l > 0$.

For any word x, there is a word $r(x) > x$ such that $g(r(x))$ has size l and contains only words strictly larger than x. Assuming otherwise, by hard-wiring $A(z)$ into M for all $z \leq x$ and for all z that are contained in one of the sets $g(y)$ with $y \leq x$, we would obtain an oracle Turing machine that again btt-autoreduces A and has norm strictly smaller than M, thus contradicting the choice of M. Let x_1 be the least word x such that $g(x)$ has size l and for $s > 1$, let

$$x_{s+1} = r(\max J_s) \quad \text{where} \quad J_s = \{x_s\} \cup g(x_s)\ .$$

By choice of r, this inductive definition yields an infinite sequence $x_1, x_2, \ldots$ such that the function $s \mapsto x_s$ is computable, the sets J_s all have size $l + 1$, and any element of J_s is less than any element in J_{s+1}.

Consider an arbitrary index s. Then $h(x_s)$ is not constant because $g(x_s)$ has size $l > 0$ and M is normalized. Thus there is an assignment on $g(x_s)$ such that $h(x_s)$ evaluates to a value different from $\perp$. Let α be the least such assignment, and let i_s be the value obtained by applying the truth-table $h(x_s)$ to α. Extend α to an assignment α_s on the set J_s where $\alpha_s(x_s) = 1 - i_s$. Observe that J_s and α_s can be computed from s.

We construct now a computable betting strategy b that succeeds on A. The construction is based on the observation that for all s, the restriction of A to J_s differs from α_s. Otherwise, $M(A, x_s) = i_s$ would differ from $A(j_s) = 1 - i_s$ and M would not autoreduce A.

The betting strategy b can be viewed as working in stages $s = 1, 2, \ldots$. The bets of stage s are all on places in J_s and use the capital accumulated until the beginning of stage s for betting against the event that the restriction of the unknown sequence to J_s is equal to α_s. Formally, the bets at stage s are performed according to the

strategy described in Remark 3.6 where $\Theta = \Theta_s$ contains all assignments on J_s that differ from α_s. The size of J_s is $l+1$; hence there are 2^{l+1} assignments to J_s, and Θ_s contains a fraction of

$$\delta = \frac{2^{l+1}-1}{2^{l+1}}$$

of all assignments on J_s. As a consequence, if the unknown sequence is indeed A, then the capital increases by the constant factor $1/\delta > 1$ during each stage s; hence b succeeds on A.

(b) Fix any sequence A that is i.o. p-btt-autoreducible. Again it suffices to show that A is not p-random, i.e., that there is a betting strategy b that is computable in polynomial time and succeeds on A. The ideas underlying the construction of b are similar to the ones used in the proof of assertion (a). The remaining differences relate to the fact that b now has to be computable in polynomial time and hence cannot perform an essentially unbounded search for places where the reduction does not yield $\perp$.

Fix an oracle Turing machine $M = (g, h)$ that p-btt-autoreduces A. For the scope of this proof, given a word w, we write x_w for $s_{|w|}$; i.e., if w is viewed as prefix of a sequence X, then x_w is the first word y such that $X(y)$ is not encoded into w. Furthermore, we write m_w and n_w for the length of w and of x_w. For any word w, let M_w be defined by

$$M_w(X, x) = M(\langle X, w\rangle, x);$$

i.e., in order to obtain M_w, the word w is hard-wired into M by overwriting the length m_w prefix of the oracle by w. For all words w, let

$$J(w) = \{x_w\} \cup \{z : z \in g(x_w) \text{ and } x_w < z\}.$$

For the scope of this proof, call a word w *promising* if $M_w(X, x_w) \neq \perp$ for some sequence X. For any promising word w, among all such sequences X let $X(w)$ be the one such that the restriction of X to $J(w) \setminus \{x_w\}$ is minimum, and let γ_w be the corresponding restriction. Let $i(w) = M_w(X(w), x_w)$, and extend γ_w to an assignment α_w on $J(w)$ by letting $\alpha_w(x_w) = 1 - i(w)$. Similar to the proof of the first assertion we can argue that for any promising prefix w of A, the restriction of A to $J(w)$ differs from α_w because otherwise M did not i.o. btt-autoreduce A.

We construct now a betting strategy b that is computable in polynomial time and succeeds on A. Define a partition $I_0, I_1, \ldots$ of the set of all words by

$$I_s := \{x : d(s) \leq |x| < d(s+1)\} \quad \text{where } d(0) = 1,\ \ d(s+1) = 2^{d(s)}.$$

There are infinitely many promising prefixes of A because in particular any prefix w of A where $M(A, x_w) \neq \perp$ is promising. In what follows we assume that there are infinitely many promising prefixes w of A such that x_w is contained in one of the even intervals $I_0, I_2, \ldots$, and we omit the virtually identical considerations for the symmetrical case where there are infinitely many promising prefixes w where x_w is contained in an odd interval.

The betting strategy b works similarly to the one used in the proof of assertion (a), and we leave the details of its construction and verification to the reader. By definition of the intervals I_i, we can pick an index s_0 such that for all $s > s_0$ and for all w with x_w in I_{2s}, the set $J(w)$ is contained in the double interval $I_{2s} \cup I_{2s+1}$. During stage s,

the betting strategy b bets on words in this double interval as follows. If $s \leq s_0$ or if there is no promising prefix v of the current input w such that x_v is in I_{2s}, abstain from betting. Otherwise, let v_s be the shortest promising prefix of w such that x_v is in I_{2s}, and bet against the event that the restriction of the unknown sequence to $J(v_s)$ is equal to α_{v_s}. In case the unknown sequence is indeed A, there are infinitely many stages where the otherwise case applies, and, during each such stage, the capital increases at least by some constant positive factor. □

The proof of Theorem 5.4 actually shows that any i.o. p-btt-autoreducible sequence can be covered by an m^2-martingale. By standard techniques [6], one can construct an m^3-martingale that covers all sequences that are covered by an m^2-martingale; hence the class of i.o. p-btt-autoreducible sequences has measure 0 with respect to betting strategies that are computable in polynomial time.

REMARK 5.5. *Theorem* 5.4 *extends by essentially the same proof to truth-table reducibilities that may ask an arbitrary number of queries that are strictly less than the current input with respect to length-lexicographical ordering plus a constant number of strictly larger queries.*

6. A sharp bound on the density of guessed bits. From Theorem 5.1 we know that every random sequence is i.o. autoreducible. An interesting problem involves finding lower and upper bounds to the frequency at which computable autoreductions may yield bits of a random sequence.

DEFINITION 6.1. *For all $m > 0$, the* density $\rho(E, m)$ *of a set E up to m is defined by*

$$\rho(E, m) \;=\; \frac{|E \cap \{\mathrm{s}_0 \ldots \mathrm{s}_{m-1}\}|}{m}\,. \tag{6.1}$$

An oracle Turing machine M i.o. T-autoreduces a sequence X with density $r(m)$ *if M i.o. T-autoreduces X on a set E such that $\rho(E, m) \geq r(m)$ for all $m > 0$ (i.e., the density of the set of words x such that M guesses $X(x)$ is always at least $r(m)$).*

A sequence A is called i.o. T-autoreducible with density $r(m)$ *if there is some oracle Turing machine that i.o. T-autoreduces A with density $r(m)$. Concepts like* i.o. tt-autoreducible with density $r(m)$ *are defined in the same manner.*

Thus, autoreducibility with density $r(m)$ measures the density of the guessed bits of an autoreduced sequence A in the sense that the ratio of guessed bits to bits considered is at least $r(m)$. It should be noted that the concept of density is sort of inverse to the previous concept of *rate* [19], where an autoreduction has rate $r(m)$ if the mth place guessed is not larger than $\mathrm{s}_{r(m)}$.

We now study the question of what is the highest density a reducibility may achieve with random sequences of a certain type. A lower bound on the achievable density is easily obtained from the proof of Theorem 5.1, by noting that the autoreductions employed there obtain densities that depend on the length of the codewords in the employed error-correcting codes. In Example 6.2, we state corresponding bounds that are obtained by choosing the lengths of the codewords according to specific converging sums. Afterwards, in Theorem 6.6, we improve on this bound by elaborating the proof of Theorem 5.1.

EXAMPLE 6.2. *Fix any rec-random sequence R. Let $j_1, j_2, \ldots$ be any nondecreasing computable sequence such that*

$$\sum_{k=0}^{\infty} \frac{1}{l_k} \;<\; \infty \qquad \text{where } l_k = 2^{j_k} - 1 > 0.$$

Then also the sum over the $\frac{1}{l_k+1}$ *converges; hence as in the proof of Theorem* 5.1 *we can find perfect one-error-correcting codes* C_k *of codeword length* l_k *such that the sequence* R *can be written as* $w_1 w_2 \dots$ *where each word* w_k *has length* l_k *and almost all* w_k *are not in* C_k*. Moreover, by hard-wiring finitely many bits of* R *plus applying the solution of the hat problem to the words* w_k*, we obtain an autoreduction of* R *which guesses exactly one bit of each of the words* w_k*. In this situation, let* E *be the set of the bits that are guessed correctly, and let* $\rho(E, m)$ *be the density of* E *defined in* (6.1)*.*

We aim at deriving upper and lower bounds for $\rho(E, m)$*. If we let* $i(m)$ *be the maximum index* i *such that* $a_i = l_1 + \cdots + l_i \leq m$*, then by construction for all* m *we have*

$$\frac{i(m)}{m} \leq \rho(E, m) \leq \frac{i(m)+1}{m}\ ; \tag{6.2}$$

i.e., in order to bound $\rho(E, m)$ *it suffices to bound* $i(m)$*.*

If, as in the proof of Theorem 6.6*, we let* $l_k = 2^k - 1$*, then accordingly* a_i *is equal to* $2^{i+1} - i - 2$*, and* $i(m)$ *and* $\rho(E, m)$ *are approximately* $\log m$ *and* $\log m/m$*, respectively. In order to improve on this density, we might try to use values for* l_k *that grow slower. If we fix* $\delta > 1$*, the sum* $\sum 1/(k \log^\delta k)$ *converges; hence we might choose* l_k *as the least number of the form* $2^j - 1$ *that is greater than or equal to* $k \log^\delta k$*. Some easy calculations, which are left to the reader, show that for almost all* i*, we have* $i^2 \leq a_i \leq i^{2.001}$*; thus, by* $a_{i(m)} \leq m \leq a_{i(m)+1}$*,*

$$i(m)^2 \leq m \leq (i(m)+1)^{2.001} \leq i(m)^{2.002}\ ; \quad \textit{hence} \quad m^{\frac{1}{2.002}} \leq i(m) \leq m^{\frac{1}{2}}\ .$$

By (6.2)*, for almost all* m *the density* $\rho(E, m)$ *is between* $m^{\frac{1}{2.002}}/m$ *and* $m^{\frac{1}{2}}/m$*; i.e., the achieved density is approximately equal to* $1/\sqrt{m}$*.*

In what follows, we prove something much stronger than the result from Example 6.2, namely, that for any computable function r that goes nonascendingly to 0, any rec-random sequence is i.o. truth-table-autoreducible with density $r(m)$. This result is consummately complemented by our next theorem, which shows that rec-random sequences are never i.o. truth-table-autoreducible with positive constant density (i.e., with density $r(m) = \varepsilon m$ for some $\varepsilon > 0$) and that a similar assertion holds with respect to Martin-Löf random sequences and i.o. Turing autoreducibility.

THEOREM 6.3.

(a) *No rec-random sequence is i.o. tt-autoreducible with positive constant density.*

(b) *No Martin-Löf random sequence is i.o. T-autoreducible with positive constant density.*

Proof. Assertions (a) and (b) are proved by showing that if a sequence is i.o. autoreducible with constant density, then the sequence cannot be random. In both cases, the argument relies on Claims 1 and 2 below.

Fix an oracle Turing machine M and a rational $\varepsilon_0 > 0$. We want to show that if M i.o. autoreduces a sequence with density $\varepsilon_0 m$, then this sequence cannot be random; i.e., an appropriate betting strategy succeeds on this sequence. Recall that the cylinder generated by a word w is the class $w\{0,1\}^\infty$ of all sequences that extend w. We argue that for any w, the fraction of sequences in this cylinder that are i.o. autoreduced by M with density ε_0 is bounded by a constant $\delta < 1$, which does not depend on w. Let $\varepsilon = \varepsilon_0/2$, and for any $m > 0$ let

$$i(m) = \left\lceil \frac{m}{\varepsilon} \right\rceil\ , \qquad I(m) = \{\mathrm{s}_m, \dots, \mathrm{s}_{m+i(m)-1}\}; \tag{6.3}$$

that is, assuming $|w| = m$, the interval $I(m)$ contains the first $i(m)$ words that are not in the domain of w. For any sequence X and any finite set I, let

$$\begin{aligned} \text{correct}(X, I) &= |\{x \in I : M(X, x) = X(x)\}|, \\ \text{incorrect}(X, I) &= |\{x \in I : M(X, x) = 1 - X(x)\}|. \end{aligned}$$

That is, for all inputs x in I such that $M(X, x)$ is defined and differs from $\perp$, we count for how many of these inputs the guess $M(X, x)$ is correct and for how many it is incorrect. In the remainder of this proof and with M and ε understood from the context, we say a sequence X is consistent with a word w of length m if

(i) X is an extension of w,
(ii) for all x in $I(m)$, the value $M(X, x)$ is defined and is computed without querying the oracle at place x,
(iii) $\text{incorrect}(X, I(m)) = 0$,
(iv) $\text{correct}(X, I(m)) \geq \varepsilon |I(m)|$.

CLAIM 1. *If M i.o. T-autoreduces a sequence with density ε_0, then this sequence is consistent with any prefix of itself.*

Proof. Fix any sequence A that satisfies the assumption of the claim, and consider any prefix w of A. Then the conditions (i), (ii), and (iii) are satisfied by definition (recall that $M(A, x)$ is always defined in case M i.o. T-autoreduces A). If condition (iv) was false, then contrary to our assumption on A the oracle Turing machine M with oracle A would guess in the interval $I(m)$ and among the m smaller words in total strictly less than

$$m + \varepsilon |I(m)| \;\leq\; 2\,\varepsilon |I(m)| \;=\; \varepsilon_0 |I(m)| \tag{6.4}$$

bits, where the relations in (6.4) hold by (6.3) and by choice of ε. □

For any word w, let the sequence X_w be an extension of w that is obtained by the chance experiment where the bits of X_w that are not already determined by w are obtained by independent tosses of a fair coin. Let $\delta = 1/(1+\varepsilon)$, and observe that $\delta < 1$ due to $\varepsilon > 0$.

CLAIM 2. *For any word w, the probability that X_w is consistent with w is at most δ.*

Proof. Fix any word w of length m. The key observation in the proof of Claim 2 is the following. If we examine for all inputs x in $I(m)$ such that the value $M(X_w, x)$ is in $\{0, 1\}$ at all, whether this value is a correct guess in the sense that it agrees with $X_w(x)$, then the expected number of correct and incorrect guesses is the same; i.e.,

$$\mathbf{E}[\text{correct}(X_w, I(m))] \;=\; \mathbf{E}[\text{incorrect}(X_w, I(m))]\,. \tag{6.5}$$

For a proof, first consider a single input x in $I(m)$. The assignment to X_w at x and at the places different from x are stochastically independent; hence by (ii) the same holds for $X_w(x)$ and $M(X_w, x)$. Furthermore, since $X_w(x)$ is chosen uniformly from $\{0, 1\}$, it follows that the probability for a correct and for an incorrect answer at x are both exactly half of the probability that $M(X_w, x)$ is in $\{0, 1\}$. Hence also the expected number of correct and incorrect answers at x coincide, and (6.5) follows by linearity of expectation. Now we distinguish two cases.

Case I: $\mathbf{E}[\text{correct}(X_w, I(m))] \leq \delta\varepsilon |I(m)|$.

The random variable $\text{correct}(X_w, I(m))$ is nonnegative; hence the case assumption implies that the probability that $\text{correct}(X_w, I(m))$ is at least $\varepsilon |I(m)|$ is at most δ.

So also the probability that X_w is consistent with w is at most δ by condition (iv) in the definition of consistency.

Case II: $\mathbf{E}[\text{correct}(X_w, I(m))] > \delta\varepsilon|I(m)|$.

By (6.5), we also have $\mathbf{E}[\text{incorrect}(X_w, I(m))] > \delta\varepsilon|I(m)|$. The latter implies that the probability that $\text{incorrect}(X_w, I(m))$ is strictly larger than 0 is at least $\delta\varepsilon$ because by definition, the random variable $\text{incorrect}(X_w, I(m))$ is bounded by $|I(m)|$. Due to condition (iii), the probability that X_w is consistent is then at most

$$1 - \delta\varepsilon = 1 - \frac{\varepsilon}{1+\varepsilon} = \frac{1}{1+\varepsilon} = \delta\ . \tag{6.6}$$

The assertion of Claim 2 holds in both cases; hence the claim follows. □

Proof of (a). We can assume that M is in fact a tt-reduction, say, $M = (g, h)$. Fix any sequence A such that M i.o. tt-autoreduces A with density $r(m) = \varepsilon_0 m$. It suffices to show that A is not rec-random, and this is done by constructing a computable betting strategy that succeeds on A.

The set of all words is partitioned into consecutive intervals $J_0, J_1, \ldots$, where the cardinality of J_i is denoted by l_i (i.e., J_0 contains the first l_0 words, J_1 the next l_1 words, and so on). The J_i are defined via specifying the l_i inductively. For all i, let $m_i = l_0 + l_1 + \cdots + l_i$. Let $l_0 = 1$, and for all i choose l_{i+1} so large that the union of the intervals J_0 through J_{i+1} contains $I(m_i)$ as well as the query sets $g(x)$ for all x in $I(m_i)$. This way, for any word w of length m_i, the consistency with w of any sequence X that extends w depends only on the restriction of X to the interval J_{i+1}. Call an assignment on J_{i+1} consistent with such a word w if the assignment is the restriction of a sequence that is consistent with w. By Claims 1 and 2, for any given word w of length m_i the following assertions hold with respect to the assignments on J_{i+1}.

- If w is a prefix of A, then the assignment obtained by restricting A to J_{i+1} is consistent with w.
- The assignments that are consistent with w comprise a fraction of at most δ of all assignments.

Now consider the betting strategy that for each interval J_{i+1} uses the capital accumulated up to the first element of the interval in order to bet according to Remark 3.6 against all assignments on this interval that are not consistent with the already seen length m_i prefix of the unknown sequence. By the preceding discussion, in case the unknown sequence is indeed A, then on each interval b increases its capital at least by the constant factor $1/\delta > 1$; i.e., in this case b succeeds on A. Furthermore, the betting strategy b is computable since consistency of assignments can be decided effectively.

Proof of (b). For a given word w, let $\boldsymbol{E}_w$ be the class of all sequences that are consistent with w. For a word w of length $m \geq 1$, call a word u a consistent extension of w if for some sequence U,

- w is a prefix of u and u is a prefix of U, while U is consistent with w;
- the domain of u contains $I(m)$, and all queries that are made while computing $M(U, x)$ for any x in $I(m)$;
- for all x in $I(m)$, the computation of $M(U, x)$ terminates in at most $|u|$ steps.

Let $E(w)$ be the set of minimum consistent extensions of w (i.e., $E(w)$ contains any word if the word itself but none of its prefixes is a consistent extension of w). Then for any nonempty word w,

(i) $\boldsymbol{E}_w$ is the disjoint union of the cylinders generated by words in $E(w)$,

(ii) $\boldsymbol{E}_w$ comprises at most a fraction of δ of the cylinder generated by w,
(iii) $E(w)$ is computably enumerable in w.

Assertions (ii) and (iii) hold, respectively, due to Claim 2 and because for given w and u, it can be effectively checked whether u is a consistent extension of w. Concerning assertion (i), first observe that for any sequence in $\boldsymbol{E}_w$ all prefixes of sufficient length are consistent extensions of w. On the other hand, for any consistent extension u of w, the cylinder generated by u is a subclass of $\boldsymbol{E}_w$ because any sequence that extends u is consistent with w. Furthermore, by the minimality condition in the definition of $E(w)$, the words in $E(w)$ are mutually incomparable; hence the cylinders generated by these words are mutually disjoint.

Let $\boldsymbol{C}$ be the class of all sequences that are i.o. T-autoreduced by M with density ε_0. We conclude the proof of assertion (b) by constructing a Martin-Löf null cover for $\boldsymbol{C}$. Let $V_0 = \{0, 1\}$, and for all $i \geq 0$, let

$$V_{i+1} \quad = \quad \bigcup_{w \in V_i} E(w) \, .$$

Then the sets V_i are uniformly computably enumerable; i.e., $V_i = \mathrm{W}_{h(i)}$ for some computable function h. Using (i) and the fact that any sequence in this class is consistent with all of its prefixes, an induction argument shows that for all i the class $\boldsymbol{C}$ is contained in the union of the cylinders generated by the words in $\mathrm{W}_{h(i)}$. Furthermore, another induction argument, which uses (ii) in the induction step, shows that the union of the cylinders generated by $\mathrm{W}_{h(i)}$ has measure of at most δ^i. So if we fix a constant c such that δ^c is at most $1/2$, then $\mathrm{W}_{h(ci)}$ has measure at most $1/2^i$. In summary, $\mathrm{W}_{h(c0)}, \mathrm{W}_{h(c1)}, \ldots$ is a Martin-Löf null cover for $\boldsymbol{C}$. □

The concept of density can be relativized to an infinite subset Z of all words; i.e., we might say an oracle Turing machine M i.o. T-autoreduces a given sequence with *density $r(m)$ relative to Z* if the fraction of guessed bits among the first m words in Z is always at least $r(m)$. By a straightforward adaptation of the proof of Theorem 6.3, it is possible to demonstrate Corollary 6.4, from which Corollary 6.5 can be obtained as a special case.

Corollary 6.4.

(a) *No rec-random sequence is i.o. tt-autoreducible with positive constant density relative to an infinite computable set.*
(b) *No Martin-Löf random sequence is i.o. T-autoreducible with positive constant density relative to an infinite computable set.*

Corollary 6.5.

(a) *No rec-random sequence is i.o. tt-autoreducible on a computable set.*
(b) *No Martin-Löf random sequence is i.o. T-autoreducible on a computable set.*

Trakhtenbrot [44] observed that no Kolmogorov–Loveland stochastic sequence [27, 32] is T-autoreducible, and his argument easily extends to i.o. T-autoreducibility on a computable set; from the latter, assertion (b) in Corollary 6.5 is immediate, because the Martin-Löf random sequences form a proper subclass of the Kolmogorov–Loveland stochastic sequences.

The "negative" assertions in Theorem 6.3 and Corollaries 6.4 and 6.5 are complemented by the two following "positive" assertions due to Merkle and Mihailović [34]. There are rec-random sequences that are weak truth-table-autoreducible. There are Martin-Löf random sequences that are self-reducible with respect to the reducibility *being computably enumerable in*, where self-reducible means autoreducible while asking only queries that are less than the current input with respect to length-lexicographical ordering.

Apparently, the arguments used in this section do not carry over to show that p-random sequences cannot be p-T- or p-tt-autoreducible; in fact, the latter assertion relates to major open problems in complexity theory. These relations are implicit in the work of Buhrman et al. [16], from which, among others, the two following implications are immediate. First, if there are p-random sequences that are p-tt-autoreducible, then the complexity classes **MA** and **EXP** are the same. Second, if no p-random sequence is p-tt-autoreducible, then the complexity classes **BPP** and **EXP** differ. The first implication is immediate by the result of Buhrman et al. that **MA** $\neq$ **EXP** implies that the p-tt-autoreducible sequences have measure 0 with respect to polynomial-time computable betting strategies. For a proof of the second implication assume that **BPP** = **EXP**. By a result of Allender and Strauss [1], any p-random sequence is p-tt-complete for **BPP**. But **EXP** contains p-random sequences; hence by our assumption some p-random sequence is p-tt-complete for **EXP**. The assertion now follows by the result of Buhrman et al. that **BPP** = **EXP** implies that all p-tt-complete sequences for **EXP** are p-tt-autoreducible.

Recall the negative result on constant bounds in Theorem 6.3; i.e., a rec-random sequence cannot be i.o. tt-autoreduced with constant positive density. This result is essentially matched by Theorem 6.6, which states that for any computable, rational-valued function r that goes nonascendingly to 0, every rec-random sequence is i.o. tt-autoreducible with density $r(m)$.

THEOREM 6.6. *Let $g\colon \mathbb{N} \to \mathbb{N}$ be any computable function that is unbounded and nondecreasing. Then every rec-random sequence is i.o. tt-autoreducible with density $r(m) = 1/g(m)$.*

Proof. Fix any rec-random sequence R. It suffices to show that R is i.o. tt-autoreducible with density $r(m)$ by some oracle Turing machine M. For every $k > 0$, let

$$l_k = 2^k - 1 \, .$$

For further use, fix perfect one-error-correcting codes C_k of codeword length l_k such that given x, we can decide effectively whether x is in one of the codes C_k. The function g is computable and unbounded; thus we can define a computable sequence of numbers $t_0 < t_1 < \cdots$, where $g(t_k) \geq l_{k+1}$, and hence

$$r(t_k) \;\leq\; \frac{1}{l_{k+1}} \, .$$

For every $k > 0$, partition the set of all words into consecutive intervals $J_k^1, J_k^2, \ldots$ of length l_k; i.e., J_k^1 contains the first l_k words, J_k^2 contains the next l_k words, and so on. Let c_k be minimum such that the $(t_k{+}1)$th word s_{t_k} is in $J_k^{c_k}$, and let

$$H_k = J_k^2 \cup \cdots \cup J_k^{c_k} \, .$$

We construct M from oracle Turing machines $M_1, M_2, \ldots$, which we refer to as modules. Intuitively, module k is meant to ensure density $1/l_k$ in the interval between s_0 and s_{t_k}. Before defining the modules, we describe their properties and how they are combined to form M.

Module k never queries any place outside the set $J_k^1 \cup H_k$ and outputs $\perp$ on all inputs that are not in H_k. Furthermore, on oracle R, module k never makes a wrong guess and guesses exactly one bit in each of the intervals J_k^2 through $J_k^{c_k}$. In addition, we will ensure that the M_k are uniformly effective in the sense that there is an oracle

Turing machine M_0 such that the values $M_k(x, X)$ and $M_0(\langle x, k\rangle, X)$ are always either both undefined or both defined and identical.

Then M is obtained by running the modules in parallel as follows. For input x, if x is equal to s_0 or s_1, then M just outputs $R(x)$. Otherwise, M determines the finitely many k such that $x \in H_k$ and simulates the corresponding modules with input x and the current oracle. If any of these modules outputs a value different from $\perp$, then M outputs the value output by the least such module; otherwise, M outputs $\perp$. Assuming the properties of the modules stated so far, we can already prove that M works as required.

Claim 1. *The oracle Turing machine M i.o. tt-autoreduces the sequence R with density $r(m)$.*

Proof. From the module assumptions, it is immediate that M is computable and that M queries its oracle nonadaptively and never queries the oracle at the current input. Moreover, on oracle R, all guesses of the modules and hence all guesses of M are correct.

Let E be the set of all inputs x such that $M(R, x)$ differs from $\perp$. Fix k and assume that m is in J_k^1 through $J_k^{c_k}$. Then we have

$$\rho(E, m) \geq 1/l_k \ . \tag{6.7}$$

This follows because M guesses the first two bits, while module k and hence M guess at least one bit in every interval except the first one, where the intervals have length l_k. So if m is in interval J_k^j, up to m there are at most jl_k words and at least j guesses; hence the density up to m is at least $1/l_k$, and (6.7) holds.

In order to prove $\rho(E, m) \geq r(m)$, fix any m and choose k such that $t_{k-1} \leq m < t_k$. Then we have

$$\rho(E, m) \geq \frac{1}{l_k} \geq r(t_{k-1}) \geq r(m) \ , \tag{6.8}$$

where the inequalities follow because m is in J_k^1 through $J_k^{c_k}$, and hence (6.7) applies, by choice of t_{k-1}, and since r is nonascending. ∎

In order to construct modules that have the required properties, let w_k^j be the restriction of R to J_k^j; i.e.,

$$R = w_k^1 w_k^2 \ldots, \quad \text{where } |w_k^j| = l_k, \text{ and let} \quad w_k = w_k^1 \oplus \cdots \oplus w_k^{c_k} \ ,$$

where $\oplus$ is bit-wise exclusive-or. The idea underlying the construction of the modules is as follows. The code words in C_k comprise such a small fraction of all words of length l_k that in case infinitely many words w_k were in C_k, there would be a computable betting strategy that succeeds on R. But R is assumed to be rec-random; hence w_k is not in C_k for almost all k. So in order to guess bits of w_k and then also of w_k^1 through $w_k^{c_k}$, we can apply the solution to the hat problem described in Remark 2.6.

Claim 2. *For almost all k, w_k is not in C_k.*

Proof. Suppose we bet on the bits of an unknown sequence X. Similar to the definition of w_k, let v_k^j be the restriction of X to the interval J_k^j, and let v_k be equal to $v_k^1 \oplus \cdots \oplus v_k^{c_k}$. Recall that C_k contains a fraction of $a_k = 1/2^k$ of all words of length l_k, and observe that the mapping

$$o_k \colon u \mapsto v_k^1 \oplus \cdots \oplus v_k^{c_k - 1} \oplus u$$

is a bijection of $\{0,1\}^{l_k}$. Thus o_k maps just a fraction of a_k of all length l_k words to C_k. When betting on the places in $J_k^{c_k}$, we have already seen v_k^1 through $v_k^{c_k-1}$. Under the assumption that v_k is in C_k, we can exclude all but a fraction of $1/a_k$ of the possible assignments on $J_k^{c_k}$, and hence, by betting in favor of these assignments according to Remark 3.6, we can increase our capital by a factor of $1/a_k$ in case the assumption is true.

Now consider the following betting strategy. For every k, a portion of a_k of the initial capital 1 is exclusively used for bets on the the interval $J_k^{c_k}$. On each such interval, the bets are in favor of the a_k-fraction of assignments that o_k maps to C_k. Then the capital a_k increases to 1 for all k such that v_k is in C_k, and consequently the betting strategy succeeds on any sequence such that the latter occurs for infinitely many k. But no computable betting strategy can succeed on the rec-random sequence R; hence Claim 2 follows. We leave it to the reader to show that this strategy is indeed computable. Observe in this connection that each word is contained in at most finitely many intervals of the form $J_k^{c_k}$, and consequently at most finitely many of the strategies related to these intervals might act in parallel. □

It remains to construct the modules. By Claim 2, fix k_0 such that w_k is not in C_k for all $k > k_0$. First assume $k \leq k_0$. Consider the least elements of the intervals J_k^2 through $J_k^{c_k}$ and for all these x, hard-wire $R(x)$ into module k. On all these x, module k outputs $R(x)$, while the module outputs $\perp$ on all other inputs.

Next assume $k > k_0$. On inputs that are not in H_k, module k simply outputs $\perp$. Now consider an input x in H_k, and suppose that

$$x \text{ is element } i_0 \text{ of interval } J_k^{j_0}, \quad 0 \leq i_0 < l_k, \;\; 2 \leq j_0 \leq c_k \; .$$

For the given oracle X, define v_k and the v_k^j as in the proof of Claim 2; i.e., v_k^j is the restriction of X to J_k^j, and v_k is the bit-wise exclusive-or of the v_k^j. Then on input x, module k queries its oracle at the remaining words in $J_k^1 \cup H_k$; i.e., the module obtains all bits of the words

$$v_k^j = v_k^j(0)\, v_k^j(1) \, \ldots \, v_k^j(l_k - 1) \; , \quad j = 1, \ldots, c_k \; ,$$

except for the bit $X(x) = v_k^{j_0}(i_0)$. In order to guess this bit, the module tries to guess the bit $v_k(i_0)$. From the latter and from the already known bits $v_k^j(i_0)$ for $j \neq j_0$, the bit $v_k^{j_0}(i_0)$ can then be computed easily since $v_k(i_0)$ is the exclusive-or of the $v_k^j(i_0)$.

In order to guess $v_k(i_0)$, module k mimics the solution of the hat problem with l_k players that is given in Remark 2.6. More precisely, the module computes the remaining bits of v_k from the $v_k^j(i)$ and obtains two consistent words u_0 and u_1. In case for some r the word u_r is in C_k, the module guesses $u_{1-r}(i_0)$, while the module abstains from guessing otherwise. By assumption on k, on oracle R the word v_k is not in C_k; hence the discussion in Remark 2.6 shows that module k never guesses incorrectly and guesses exactly one bit in each of the intervals J_k^2 through $J_k^{c_k}$ (in fact, for some i, in each interval bit i is guessed). □

Acknowledgments. We especially thank Ken Rose (Santa Barbara) for his enlightening discussions on error-correcting codes. We also acknowledge helpful hints from Klaus Ambos-Spies (Heidelberg), Charles Akemann (Santa Barbara), Dieter van Melkebeek (Madison), Klaus W. Wagner (Würzburg), and anonymous referees.

REFERENCES

[1] E. Allender and M. Strauss, *Measure on small complexity classes, with applications for BPP*, in Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Los Alamitos, CA, 1994, IEEE Computer Science Press, Los Alamitos, CA, 1994, pp. 807–818.

[2] K. Ambos-Spies, *P-mitotic sets*, in Decision Problems and Complexity, Lecture Notes in Comput. Sci. 171, E. Börger, G. Hasenjaeger, and D. Rödding, eds., Springer-Verlag, Berlin, 1984, pp. 1–23.

[3] K. Ambos-Spies, H. Fleischhack, and H. Huwig, *Diagonalizations over polynomial time computable sets*, Theoret. Comput. Sci., 51 (1987), pp. 177–204.

[4] K. Ambos-Spies, *Resource bounded genericity*, in Computability, Enumerability, Unsolvability—Directions in Recursion Theory, London Math. Soc. Lecture Note Ser. 224, S. B. Cooper, T. A. Slaman, and S. S. Wainer, eds., Cambridge University Press, Cambridge, UK, 1996, pp. 1–59.

[5] K. Ambos-Spies and A. Kučera, *Randomness in computability theory*, in Computability Theory and Its Applications—Current Trends and Open Problems, Contemp. Math. 257, P. A. Cholak, S. Lempp, M. Lerman, and R. A. Shore, eds., AMS, Providence, RI, 2000, pp. 1–14.

[6] K. Ambos-Spies and E. Mayordomo, *Resource-bounded measure and randomness*, in Complexity, Logic, and Recursion Theory, A. Sorbi, ed., Dekker, New York, 1997, pp. 1–47.

[7] K. Ambos-Spies, S. A. Terwijn, and X. Zheng, *Resource bounded randomness and weakly complete problems*, Theoret. Comput. Sci., 172 (1997), pp. 195–207.

[8] A. Arslanov, *On the phenomenon of autocomputability*, in Computing: The Australasian Theory Symposium 2000, Electron. Notes Theoret. Comput. Sci. 31, Elsevier, Amsterdam, 2000, paper 31002.

[9] J. Aspnes, R. Beigel, M. L. Furst, and S. Rudich, *The expressive power of voting polynomials*, Combinatorica, 14 (1994), pp. 135–148.

[10] R. Beigel, L. Fortnow, and F. Stephan, *Infinitely-Often Autoreducible Sets*, Research report 62, Arbeitsgruppe Mathematische Logik und Theoretische Informatik, Institut für Informatik, Universität Heidelberg, Heidelberg, Germany, 2003.

[11] J. L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity*, Vol. I, 2nd ed., Springer-Verlag, Berlin, 1995.

[12] J. L. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity*, Vol. II, Springer-Verlag, Berlin, 1990.

[13] J. L. Balcázar and E. Mayordomo, *A note on genericity and bi-immunity*, in Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, MN, 1995, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 193–196.

[14] W. Blum, *Denksport für Hutträger*, DIE ZEIT 19/2001, May 3, 2001 (in German).

[15] H. Buhrman, L. Fortnow, D. van Melkebeek, and L. Torenvliet, *Separating complexity classes using autoreducibility*, SIAM J. Comput., 29 (2000), pp. 1497–1520.

[16] H. Buhrman, D. van Melkebeek, K. W. Regan, D. Sivakumar, and M. Strauss, *A generalization of resource-bounded measure, with application to the BPP vs. EXP problem*, SIAM J. Comput., 30 (2000), pp. 576–601.

[17] C. Calude, *Information and Randomness: An Algorithmic Perspective*, 2nd ed., Springer-Verlag, Berlin, 2002.

[18] R. Daley, *On the simplicity of busy beaver sets*, Z. Math. Logik Grundlag. Math., 24 (1978), pp. 207–224.

[19] T. Ebert, *Applications of Recursive Operators to Randomness and Complexity*, Ph.D. thesis, University of California at Santa Barbara, Santa Barbara, CA, 1998.

[20] T. Ebert and W. Merkle, *Autoreducibility of random sequences: A sharp bound on the density of guessed bits*, in Mathematical Foundations of Computer Science 2002, Lecture Notes in Comput. Sci. 2420, K. Diks and W. Rytter, eds., Springer-Verlag, Berlin, 2002, pp. 221–233.

[21] T. Ebert and H. Vollmer, *On the autoreducibility of random sequences*, in Mathematical Foundations of Computer Science 2000, Lecture Notes in Comput. Sci. 1893, M. Nielsen and B. Rovan, eds., Springer-Verlag, Berlin, 2000, pp. 333–342.

[22] K. H. Kim and F. W. Roush, *Applied Abstract Algebra*, John Wiley and Sons, New York, 1983.

[23] C. Jockusch and M. Paterson, *Completely autoreducible degrees*, Z. Math. Logik Grundlagen Math., 22 (1976), pp. 571–575.

[24] R. Ladner, *Mitotic recursively enumerable sets*, J. Symbolic Logic, 38 (1973), pp. 199–211.

[25] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hyper-*

cubes, Morgan Kaufmann, San Mateo, CA, 1992.
[26] H. W. Lenstra and G. Seroussi, *On hats and other covers*, in Proceedings of the IEEE International Symposium on Information Theory, Lausanne, Switzerland, 2002.
[27] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd ed., Springer-Verlag, Berlin, 1997.
[28] J. H. Lutz, *Almost everywhere high nonuniform complexity*, J. Comput. System Sci., 44 (1992), pp. 220–258.
[29] J. H. Lutz, *The quantitative structure of exponential time*, in Complexity Theory Retrospective II, L. A. Hemaspaandra and A. L. Selman, eds., Springer-Verlag, Berlin, 1997, pp. 225–260.
[30] P. Martin-Löf, *The definition of random sequences*, Information and Control, 9 (1966), pp. 602–619.
[31] E. Mayordomo, *Contributions to the Study of Resource-Bounded Measure*, Doctoral dissertation, Universitat Politècnica de Catalunya, Barcelona, Spain, 1994.
[32] W. Merkle, *The Kolmogorov-Loveland stochastic sequences are not closed under selecting subsequences*, in International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2380, P. Widmayer, F. T. Ruiz, R. Morales, M. Hennessy, S. Eidenbenz, and R. Conejo, eds., Springer-Verlag, Berlin, 2002, pp. 390–400.
[33] W. Merkle, *The complexity of stochastic sequences*, in Proceedings of the 18th IEEE Annual Conference on Computational Complexity, Aarhus, Denmark, 2003, IEEE Computer Society Press, Los Alamitos, CA, 2003, pp. 230–235.
[34] W. Merkle and N. Mihailović, *On the construction of effective random sets*, in Mathematical Foundations of Computer Science 2002, Lecture Notes in Comput. Sci. 2420, K. Diks and W. Rytter, eds., Springer-Verlag, Berlin, 2002, pp. 568–580.
[35] S. Robinson, *Why mathematicians now care about their hat color*, New York Times, April 10, 2001.
[36] S. Rudich, *private communication*, 2001.
[37] D. E. Shasha, *Crowns of the Minotaur*, Scientific American, 285 (2001), p. 94.
[38] A. N. Shiryaev, *Probability*, Springer-Verlag, Berlin, 1995.
[39] C.-P. Schnorr, *A unified approach to the definition of random sequences*, Math. Systems Theory, 5 (1971), pp. 246–258.
[40] C.-P. Schnorr, *Zufälligkeit und Wahrscheinlichkeit*, Lecture Notes in Math. 218, Springer-Verlag, Berlin, 1971 (in German).
[41] C.-P. Schnorr, *Optimal algorithms for self-reducible problems*, in International Colloquium on Automata, Languages and Programming 1976, S. Michaelson and R. Milner, eds., Edinburgh University Press, Edinburgh, UK, 1976, pp. 322–337.
[42] R. I. Soare, *Recursively Enumerable Sets and Degrees*, Perspect. Math. Logic, Springer-Verlag, Berlin, 1987.
[43] S. A. Terwijn, *Computability and Measure*, Doctoral dissertation, Universiteit van Amsterdam, Amsterdam, The Netherlands, 1998.
[44] B. A. Trakhtenbrot, *On autoreducibility*, Soviet Math. Dokl., 11 (1970), pp. 814–817.
[45] J.H. van Lint, *Introduction to Coding Theory*, 3rd ed., Springer-Verlag, Berlin, 1999.
[46] D. van Melkebeek, *Randomness and Completeness in Computational Complexity*, Lecture Notes in Comput. Sci. 1950, Springer-Verlag, Berlin, 2000.
[47] J. Ville, *Étude Critique de la Notion de Collectif*, Gauthiers-Villars, Paris, 1939.
[48] Y. Wang, *Randomness and Complexity*, Doctoral dissertation, Universität Heidelberg, Mathematische Fakultät, INF 288, Heidelberg, Germany, 1996.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1570–1585

THE QUANTUM COMMUNICATION COMPLEXITY OF SAMPLING*

ANDRIS AMBAINIS†, LEONARD J. SCHULMAN‡, AMNON TA-SHMA§, UMESH VAZIRANI¶, AND AVI WIGDERSON‖

Abstract. Sampling is an important primitive in probabilistic and quantum algorithms. In the spirit of communication complexity, given a function $f : X \times Y \to \{0,1\}$ and a probability distribution $\mathcal{D}$ over $X \times Y$, we define the sampling complexity of $(f, \mathcal{D})$ as the minimum number of bits that Alice and Bob must communicate for Alice to pick $x \in X$ and Bob to pick $y \in Y$ as well as a value z such that the resulting distribution of (x, y, z) is close to the distribution $(\mathcal{D}, f(\mathcal{D}))$.

In this paper we initiate the study of sampling complexity, in both the classical and quantum models. We give several variants of a definition. We completely characterize some of these variants and give upper and lower bounds on others. In particular, this allows us to establish an exponential gap between quantum and classical sampling complexity for the set-disjointness function.

Key words. communication complexity, quantum communication complexity, quantum information theory, set-disjointness, the log-rank conjecture in communication complexity

AMS subject classifications. 68M10, 68Q10, 68R05

DOI. 10.1137/S009753979935476

1. Introduction.

A central question in quantum information theory is the amount of information that can be encoded into n qubits. There are different ways to formulate this question and, surprisingly, they yield completely different answers. The most natural variant of this question is the maximal amount of mutual information that can exist between a classical random variable X and a classical probability distribution Y that is obtained from a short quantum encoding of X. More than two decades ago Holevo [10] proved that the mutual information can be at most the number of qubits communicated. That is, although $2^n - 1$ complex numbers are necessary to specify the state of n quantum bits, only n bits of information can be retrieved from a superposition on n quantum bits, and communicating qubits is not more useful than just communicating classical bits.

However, there is something in quantum bits that is more powerful than classical ones. The first demonstration of that was by Bennett and Wiesner [5] who showed that if the two parties share predefined entangled qubits (that are absolutely independent of the message), then Alice can communicate $2n$ classical bits to Bob using only n

*Received by the editors April 12, 1999; accepted for publication (in revised form) June 2, 2003; published electronically October 2, 2003. An earlier version of this paper appeared in the Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS), Palo Alto, CA, IEEE Computer Society, Washington, DC, 1998, pp. 342–351.

http://www.siam.org/journals/sicomp/32-6/35476.html

†Institute of Mathematics and Computer Science, University of Latvia, Raina Bulv. 29, Riga, LV-1459 Latvia (ambainis@ias.edu).

‡Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125 (schulman@caltech.edu).

§Computer Sciences, Tel-Aviv University, Tel-Aviv, Israel 69978 (amnon@post.tau.ac.il).

¶Computer Science Division, University of California, Berkeley, CA 94720-1776 (vazirani@cs.berkeley.edu). This author's work was supported in part by DARPA grant F30602-01-2-0524 and NSF ITR grant CCR-0121555.

‖Computer Science Department, The Hebrew University, Jerusalem 91904, Israel, and School of Mathematics, The Institute for Advanced Studies, Princeton, NJ (avi@cs.huji.ac.il). This author's work was supported by grant 69/96 of the Israel Science Foundation, founded by the Israel Academy for Sciences and Humanities.

communication qubits.

Another example was supplied by Ambainis et al. [3] and by Nayak [14], where Alice's task was to encode m classical bits into n qubits ($m > n$) such that Bob could choose to read any *one* of the m encoded bits of his choice (thereby possibly destroying the information about the remaining $m-1$ bits). On the positive side they showed a scheme beating Holevo's bound, but on the negative side they showed that n can be no smaller than $\Omega(m)$.

A rich hunting ground for relevant examples is the communication complexity model [21, 20]. Buhrman, Cleve, and Wigderson [6] considered the disjointness function, where Alice and Bob get two subsets x, y of $[1, \ldots, n]$, and $DISJ(x, y) = 1$ iff x and y are disjoint. It is well known that any classical probabilistic protocol must exchange a linear number of communication bits. On the other hand, they showed that the task can be carried out with only $O(\sqrt{n}\log(n))$ quantum bits. The result is based on Grover's quantum search algorithm [9]. This provided the first asymptotic separation in power between classical and quantum communication. Recently, Razborov [17] showed an $\Omega(\sqrt{n})$ lower bound on the quantum communication complexity of the problem, and Aaronson and Ambainis [1] showed that Razborov's bound is tight up to constant factors.

Buhrman, Cleve, and Wigderson [6] also gave another communication task based on the Deutsch–Jozsa problem [8], where the number of classical bits required to compute a function *with zero error* is exponentially larger than the corresponding number of quantum bits. However, there is a probabilistic protocol with a small error probability where the number of bits exchanged is as small as the number exchanged by the quantum protocol. Raz [19] showed such an exponential gap for a *partial* function even in the presence of errors. However, the result applies only for partial functions when the two players are given a promise that their inputs come from a small (in fact, tiny) set of possible inputs.

In this paper we give the first example of a communication task for a *total* function which can be carried out by transferring exponentially fewer quantum bits than classical bits even when error is allowed. We consider the problem $DISJ_k$ that is the disjointness problem on cardinality k subsets $x, y \subseteq [n]$. However, we do not consider the number of communication bits required to *compute* the function, but rather the number of communication bits required to *sample* the function. The task is the following: Alice has a cardinality k subset $S \subseteq \{1, \ldots, n\}$, and Bob must pick a uniformly random cardinality k subset $T \subseteq \{1, \ldots, n\}$ disjoint from S. We consider the case $k = \Theta(\sqrt{n})$, and give a quantum protocol in which Alice sends $O(\log n \cdot \log 1/\epsilon)$ quantum bits to Bob, enabling him to sample from a distribution which is ϵ close (in total variation distance) to the desired uniform distribution on subsets disjoint from S. We also show that any purely classical protocol for this task must involve the exchange of $\Omega(\sqrt{n})$ bits between Alice and Bob.

We observe that applying Holevo's bound to the quantum protocol yields the following corollary: Alice and Bob can sample (with a small error) two disjoint subsets of cardinality $\sqrt{n}$ such that the number of bits of information that Bob has about Alice's subset (or that Alice has about Bob's subset) is bounded by the number of qubits transmitted, which is $O(\log(n) \cdot \log(1/\epsilon))$. It is an open question whether this secrecy can be amplified so that Alice and Bob have arbitrarily small amounts of information about each other's subsets.

More generally, given a function $f : X \times Y \to \{0, 1\}$, we consider three communication complexity measures, which we now informally discuss (and formally define

in section 2):

- The usual communication complexity of f, where Alice gets input x, Bob gets input y, and we measure the number of communication bits/qubits needed to compute $f(x,y)$. We denote the classical probabilistic communication complexity by $R_\epsilon(f)$, where ϵ is the error probability, and this probability is over the random coins of Alice and Bob. The communication complexity when no error is allowed is denoted by $D(f)$. The quantum communication complexity is denoted by $Q_\epsilon(f)$.
- The communication complexity of generating the superposition of the function. Here, there is no input to the two parties, and we measure the number of qubits needed to generate the superposition $\sum_{x,y}(-1)^{f(x,y)}|x,y\rangle$, where Alice holds the X register and Bob the Y register. We call this the complexity of *generating* the function, and denote it by $\overset{\bullet}{Q}_\epsilon(f)$.
- The communication complexity of sampling values of f. Here, Alice and Bob are again given no input, and they want to sample $(x,y,z=f(x,y))$, where Alice holds x and Bob holds y. We call this the complexity of sampling the function, and denote it by $\overset{\circ}{R}_\epsilon(f)$ in the classical case and $\overset{\circ}{Q}_\epsilon(f)$ in the quantum case.

For formal definitions, see section 2. As expected, sampling is easier than generating, which in turn is easier than solving the problem on a given instance, $\overset{\circ}{Q}_\epsilon(f) \leq \overset{\bullet}{Q}_\epsilon(f) \leq Q_\epsilon(f)$. For the precise statements we prove, see Lemmas 5.1 and 5.2.

We show a tight characterization of $\overset{\bullet}{Q}_\epsilon(f)$, the complexity of generating a function. Given f, we define the matrix M_f, $M_f[x,y] = (-1)^{f(x,y)}$. We show that $\overset{\bullet}{Q}_\epsilon(f)$ relates to the best low-rank approximation of M_f, namely,

$$\overset{\bullet}{Q}_\epsilon(f) \approx \min_{A:||A-M_f||_2^2 \leq \epsilon} \log(\mathrm{rank}(A)).$$

We believe that this characterization is important by itself. From that we deduce that

$$\overset{\bullet}{Q}_\epsilon(DISJ_k) = O(\log n \cdot \log 1/\epsilon).$$

We also show, using a combinatorial lemma of Babai, Frankl, and Simon [4], that for some constant $\epsilon > 0$

$$\overset{\circ}{R}_\epsilon(DISJ_k) = \Omega(\sqrt{n}),$$

establishing an exponential gap between classical and quantum sampling. Also, as we can efficiently quantum sample (generate) the $DISJ_k$ function, we can also efficiently quantum sample (generate) the $DISJ$ function. Razborov's lower bound [17] then shows an exponential gap between quantum sampling (generating) and normal quantum communication complexity.

We conclude with a remark concerning the log-rank conjecture in communication complexity. The conjecture asks whether always $D(f) \leq Poly(\log(\mathrm{rank}(M_f)))$. Raz and Spieker [18] were the first to show a superlinear gap, and the biggest gap known today, due to Nisan and Wigderson [15], exhibits an f with $D(f) \geq \log(\mathrm{rank}(M_f))^{1.6\ldots}$ (see [16, section 2.5]). It is quite possible, for example, that $D(f) \leq \log(\mathrm{rank}(M_f))^2$. The above characterization shows that when $\epsilon = 0$, $\overset{\bullet}{Q}_0(f) = \Theta(\log(\mathrm{rank}(M_f)))$. In

fact, we show that this holds not only for quantum generating f, but also for quantum sampling f, and $\overset{\circ}{Q}_0(f) = \Theta(\log(\mathrm{rank}(M_f)))$. For the precise statement, see Theorem 8.1. This is the first example of a communication task for which the famous log-rank conjecture holds.

Furthermore, we show that zero error classical computing is almost as easy as sampling, or more precisely that $\sqrt{D(f)} \leq \overset{\circ}{R}_0(f) \leq D(f)$. We thus see that the log-rank conjecture is equivalent to the conjecture that $\overset{\circ}{R}_0(f) \leq Poly(\overset{\circ}{Q}_0(f))$, and can be cast as asking about the relative power of quantum and classical sampling in the no error case.

2. Sampling. The two-party communication complexity model, as introduced by Yao [21], consists of two players that have private inputs and wish to compute a known function that depends on both inputs. The players follow a predefined protocol and exchange communication bits until they are ready to make a decision.

In the quantum communication complexity model [20], Alice and Bob hold qubits. When the game starts, Alice holds x and Bob holds y, and so the initial superposition is simply $|x, y\rangle$. The players take turns. Suppose it is Alice's turn to play. Alice can make an arbitrary unitary transformation on her qubits and then send one or more qubits to Bob. Sending qubits does not change the overall superposition but rather changes the ownership of the qubits, allowing Bob to apply his next unitary transformation on the newly received qubits. Each player can also (partially) measure his/her qubits. By the end of the protocol, the two players have to decide on a value. If during the protocol the two players are in the system ϕ, then ϕ_{Alice} denotes the state of the subsystem of Alice's qubits, and ϕ_{Bob} is the state of the subsystem of Bob's qubits. ϕ_{Alice} and ϕ_{Bob} are usually mixed states.

The complexity of a classical (quantum) protocol is the number of bits (qubits) exchanged between the two players. We say a (quantum) protocol *computes* $f : X \times Y \mapsto \{0, 1\}$ with $\epsilon \geq 0$ error if for any input x, y the probability that the two players compute $f(x, y)$ is at least $1-\epsilon$. We denote by $R_\epsilon(f)$ ($Q_\epsilon(f)$) the complexity of the best (quantum) protocol that computes f with at most ϵ error. The deterministic complexity $D(f)$ is simply $R_0(f)$.

2.1. Sampling complexity. In the previous definitions the two players had to *compute* the right answer for a given input (x, y). A sampling protocol, however, starts with no input to the two players. Instead, by the end of the protocol, Alice holds some $x \in X$, Bob holds some $y \in Y$, and they also hold some "answer" $z \in \{0, 1\}$. We say that the protocol induces a distribution $\mathcal{P}$ on (x, y, z), where $\mathcal{P}(x, y, z)$ is the probability that x and y are sampled along with the answer z.

DEFINITION 2.1. *A classical distribution over X is a function $\mathcal{D} : X \mapsto [0, 1]$ such that (s.t.) $\sum_{x \in X} \mathcal{D}(x) = 1$. Given two distributions $\mathcal{D}_1, \mathcal{D}_2$ over X, the variational distance between them is $|\mathcal{D}_1 - \mathcal{D}_2|_1 \stackrel{\text{def}}{=} \sum_x |\mathcal{D}_1(x) - \mathcal{D}_2(x)|$.*

DEFINITION 2.2 (sampling). *Let $f : X \times Y \mapsto \{0, 1\}$, and let $\mathcal{D}$ be any distribution on $X \times Y$. We say that the protocol samples f according to $\mathcal{D}$ with ϵ error if the distribution the protocol induces on $\{(x, y, z)\}$ is ϵ close, in the total variation distance, to the distribution $(\mathcal{D}, f(\mathcal{D}))$ obtained by first picking (x, y) according to $\mathcal{D}$ and then evaluating $f(x, y)$. We denote by $\overset{\circ}{R}_\epsilon(f, \mathcal{D})$ ($\overset{\circ}{Q}_\epsilon(f, \mathcal{D})$) the number of communication bits (qubits) needed for a randomized (quantum) protocol P to sample f according to $\mathcal{D}$ with ϵ error. When $\mathcal{D}$ is the uniform distribution, we sometimes omit it.*

2.2. q–generating. In the quantum model it makes sense not only to sample the right classical distribution, but also to approximate the right quantum superposition. For example, we can ask how many communication qubits are needed for two players to generate (or approximate) the superposition $\psi = \sum_{x,y}(-1)^{\sum_i x_i y_i}|x,y\rangle$. We need to specify what is a good approximation of a superposition, and a natural choice is the so called "fidelity" measure: ϕ approximates ψ to within ϵ if $|\langle\phi|\psi\rangle| \geq 1-\epsilon$. We also allow the players to use ancillary bits.

DEFINITION 2.3 (q–generating). *We say that a quantum protocol q–generates a superposition $\psi = \sum_{x\in X, y\in Y} a_{x,y}|x,y\rangle$ to within ϵ error if it starts with no inputs to the two players and by the end of the protocol the two players compute a superposition ϕ, where ϕ_{Alice} has support in $X \otimes Ancila_X$, ϕ_{Bob} has support in $Y \otimes Ancila_Y$, and $|\langle\phi|\psi\rangle| \geq 1-\epsilon$.*

DEFINITION 2.4. *Let $f : X \times Y \mapsto \{0,1\}$ be any Boolean function and $\mu : X \times Y \mapsto C$ an l_2 distribution (i.e., $\sum_{x,y}|\mu_{x,y}|^2 = 1$). We say that a quantum protocol q–generates f according to the distribution μ with ϵ error if it q–generates the superposition $\sum_{x,y}\mu_{x,y}(-1)^{f(x,y)}|x,y\rangle$ to within ϵ error. We denote the number of communication qubits needed for this by $\overset{\bullet}{Q}_\epsilon(f,\mu)$.*

3. Preliminaries. Two superpositions that are close to each other in the fidelity norm (i.e., $|\langle\phi_1|\phi_2\rangle| \geq 1-\epsilon$) cannot be effectively distinguished. More precisely, for a superposition ϕ and a complete measurement $\mathcal{O}$ over it, let us denote by $\phi^{\mathcal{O}}$ the classical distribution (over all possible results) obtained by applying the measurement $\mathcal{O}$ over ϕ. By, e.g., Aharonov, Kitaev, and Nisan [2, Lemma 11], we have the following.

FACT 3.1 (see [2]). *For any two superpositions ϕ_1, ϕ_2 and any complete measurement $\mathcal{O}$,*

$$|\phi_1^{\mathcal{O}} - \phi_2^{\mathcal{O}}|_1 \leq 2\sqrt{1-|\langle\phi_1|\phi_2\rangle|^2}.$$

3.1. Some matrix algebra. Any normal matrix N can be diagonalized by an appropriate unitary basis change; that is, there is some unitary transformation U s.t. $UNU^\dagger$ is diagonal with the eigenvalues $\lambda_1, \ldots, \lambda_N$ on the diagonal. Singular values and the singular value decomposition theorem generalize this to arbitrary matrices. Given any (possibly nonsquare) matrix M, $MM^\dagger$ is a nonnegative matrix and hence has a complete set of nonnegative eigenvalues $\lambda_1 \geq \cdots \geq \lambda_N \geq 0$. The ith singular value, $\sigma_i(M)$, is $\sqrt{\lambda_i}$. The SVD theorem says the following.

THEOREM 3.1 (see [11, Lemma 7.3.1]). *For any matrix M there are unitary transformations U_1, U_2 s.t. $U_1 M U_2$ is diagonal with the singular values $\sigma_1(M), \ldots, \sigma_N(M)$ on the diagonal.*

Given a matrix $A = (a_{i,j})$, we define its norm $||A||_2 \stackrel{\text{def}}{=} (\sum_{i,j}|a_{i,j}^2|)^{1/2}$, i.e., $||A||_2^2 = Trace(AA^\dagger)$. The Hoffman–Wielandt theorem states the next result.

THEOREM 3.2 (see [11, Corollary 7.3.8]). *Let A and B be two matrices of the same dimensions. Then,*

$$\sum_{i=1}^{N}[\sigma_i(A) - \sigma_i(B)]^2 \leq ||B-A||_2^2.$$

Let B be an arbitrary norm one matrix, $||B||_2^2 = 1$. It follows that $\sum_i \sigma_i^2(B) = Tr(BB^\dagger) = 1$. Let $K_\epsilon(B)$ denote the number of singular values we need to take to collect $1-\epsilon$ weight; i.e., it is the first integer k such that $\sum_{i=1}^k \sigma_i^2(B) \geq 1-\epsilon$.

CLAIM 3.1. $K_\epsilon(B) = \min_{A:||A-B||_2^2 \leq \epsilon} \text{rank}(A)$.

Proof. Let us define $K'_\epsilon(B) = \min_{A:||A-B||_2^2 \le \epsilon} \operatorname{rank}(A)$.

On the one hand, say $K_\epsilon(B) = k$ and $B = U_1 D U_2$, where D is a diagonal matrix with the singular values on the diagonal. Let $\bar{D}$ be the matrix containing only the first k singular values, and $A = U_1 \bar{D} U_2$. Then A has low rank and approximates B to within ϵ. It follows that $K'_\epsilon(B) \le k = K_\epsilon(B)$.

On the other hand, say $K'_\epsilon(B) = k$ and A has rank k and $||A - B||_2^2 \le \epsilon$. It then follows by the Hoffman–Wielandt theorem that $\sum_{i=1}^N [\sigma_i(A) - \sigma_i(B)]^2 \le ||B - A||_2^2 \le \epsilon$. As A has rank k, for at least $N - k$ values i, $\sigma_i(A) = 0$. It then must follow that the squares of the $N - k$ smallest singular values of B must sum up to no more than ϵ, i.e., $K_\epsilon(B) \le K'_\epsilon(B)$. ∎

4. A tight bound on q–generating. We completely characterize the complexity of q–generating. With each superposition $\psi = \sum_{x \in X, y \in Y} a_{x,y} |x, y\rangle$ we associate a $|X| \times |Y|$ matrix $M_\psi = (a_{x,y})$. We characterize the complexity of q–generating ψ in terms of the spectrum of M_ψ. We prove the following result.

THEOREM 4.1. *For any pure state ψ and $0 \le \epsilon \le \frac{1}{2}$*

$$\lceil \log K_{2\epsilon} \rceil \le \overset{\bullet}{Q}_\epsilon(\psi) \le \lceil \log K_\epsilon \rceil,$$

where $K_\epsilon = \min_{A:||M_\psi - A||_2^2 \le \epsilon} \operatorname{rank}(A)$. Equivalently, K_ϵ is the first integer K s.t. $\sum_{i=1}^K \sigma_i^2(M_\psi) \ge 1 - \epsilon$.

4.1. The upper bound. Suppose that Alice and Bob are in a superposition $\phi = \sum_{x,y} M_{x,y} |x, y\rangle$ represented by the matrix $M = M_\phi$ (i.e., $M[x, y] = M_{x,y}$). Let us check how the matrix representation changes as Alice applies a local unitary transformation T on her qubits. The resulting superposition is

$$\begin{aligned}
(T \otimes I)\phi &= \sum_{x,y} M_{x,y} |Tx, y\rangle \\
&= \sum_{x,y} M_{x,y} \sum_z T_{z,x} |z, y\rangle \\
&= \sum_{z,y} \left(\sum_x T_{z,x} M_{x,y} \right) |z, y\rangle \\
&= \sum_{z,y} (TM)_{z,y} |z, y\rangle,
\end{aligned}$$

and so the resulting superposition is represented by TM. Similarly if *Bob* applies a local transformation T on M, the resulting superposition is represented by MT^t.

Suppose that the parties want to generate a superposition ψ represented by $M = M_\psi$. By the singular decomposition theorem (Theorem 3.1) there are unitary transformations U_1, U_2 s.t. $U_1^{-1} M U_2^{-1}$ is the diagonal matrix D with $\sigma_1(M), \ldots, \sigma_N(M)$ on the diagonal. Let $\Lambda = \{w_i | i = 1, \ldots, K\}$ be the set of the first $K = K_\epsilon$ ("heavy") eigenvectors. Let Π be the projection operator onto Λ, i.e., $\Pi[x, y]$ is 1 if $x = y$ and $1 \le x \le K$, and zero otherwise. The protocol is the following:

- Alice prepares the superposition $D\Pi$ (which is simply the superposition $c \cdot \sum_{i=1}^K \sigma_i(M) |i, i\rangle$, where $c = 1/\sqrt{\sum_{i=1}^K \sigma_i^2(M)}$, and notice that $1 \le c \le 1/\sqrt{1 - \epsilon}$) and sends the Y qubits to Bob.
- Alice applies the transformation U_1 on her qubits, and Bob applies the transformation U_2^t on his qubits.

Say that the resulting superposition is ϕ and its matrix is M_ϕ. We know that $M_\phi = cU_1 D\Pi U_2$. We have

$$\begin{aligned}||M_\phi - M_\psi||_2^2 &= ||cU_1 D\Pi U_2 - U_1 D U_2||_2^2 \\ &= ||U_1(cD\Pi - D)U_2||_2^2 \\ &= ||cD\Pi - D||_2^2 \\ &= \sum_{i>K} \sigma_i^2(M) + (c-1)^2 \sum_{i=1}^{K} \sigma_i^2(M) \\ &\leq \epsilon + \frac{(c-1)^2}{c^2} \leq \epsilon + \epsilon^2 \leq 2\epsilon.\end{aligned}$$

The third equality is due to the fact that for every matrix X and unitary matrix U, $||UX||_2^2 = \langle UX|UX\rangle = \langle X|X\rangle = ||X||_2^2$. To see the last inequality, remember that $c \leq \frac{1}{\sqrt{1-\epsilon}}$, and therefore $\frac{c-1}{c} \leq \frac{1/\sqrt{1-\epsilon}-1}{1/\sqrt{1-\epsilon}} = 1 - \sqrt{1-\epsilon} \leq \epsilon$.

To finish the proof of the upper bound of Theorem 4.1, we claim the following.

CLAIM 4.1. $|\langle\phi|\psi\rangle| \geq 1 - \epsilon$.

Proof. We treat the matrices M_ϕ, M_ψ as vectors of length $|X| \cdot |Y|$ and notice that $\langle M_\phi | M_\psi\rangle = \langle\phi|\psi\rangle$ by the way the matrices M_ϕ, M_ψ were defined.

Also, since $(U_1^{-1} \otimes U_2^{-1})\psi = \sum_i \sigma_i |i,i\rangle$ and $(U_1^{-1} \otimes U_2^{-1})\phi = c\sum_{i\in\Lambda} \sigma_i |i,i\rangle$, it follows that $\langle\phi|\psi\rangle$ is real. We then see that

$$\begin{aligned}||M_\phi - M_\psi||_2^2 &= \langle M_\phi - M_\psi | M_\phi - M_\psi\rangle \\ &= \langle M_\phi|M_\phi\rangle + \langle M_\psi|M_\psi\rangle - 2\langle M_\phi|M_\psi\rangle.\end{aligned}$$

However, $||M_\phi||_2 = ||M_\psi||_2 = 1$, and so

$$||M_\phi - M_\psi||_2^2 = 2(1 - \langle\phi|\psi\rangle).$$

Plugging $||M_\phi - M_\psi||_2^2 \leq 2\epsilon$ into this, we get $\langle\phi|\psi\rangle \geq 1 - \epsilon$ as desired. ∎

4.2. The lower bound. The lower bound idea is an extension of an idea from Kremer's thesis [12], where it is attributed to Yao. We first show that the outcome of any quantum protocol that uses only l communication qubits can be described as a linear combination of up to 2^l product superpositions (we give a precise statement soon). We use this to show that a quantum sampling protocol is actually a low rank approximation of M_ψ. We then use the Hoffman–Wielandt inequality to derive a lower bound on l.

CLAIM 4.2 (see [12]). *Suppose that P is a quantum protocol that uses l communication qubits, starts with no input, and computes the superposition ϕ. Further assume that the last qubit communicated is w_l. Then $\phi = \sum_{w\in\{0,1\}^l} |A(w), w_l, B(w)\rangle$, where A and B depend only on w.*

Proof. The proof is by induction on l. The case $l = 0$ is immediate. Suppose it is true for l; let us prove it for $l+1$. Assume after l steps that the two parties are in the superposition $\sum_{w\in\{0,1\}^l} |A(w), w_l, B(w)\rangle$ and w.l.o.g. it is now Alice's turn to play. Alice first does some unitary transformation on her qubits, which results in $\sum_{w\in\{0,1\}^l} |A'(w_1,\ldots,w_l), B(w_1,\ldots,w_l)\rangle$. Then she sends the qubit z to Bob. For every $w_1,\ldots,w_l$ we can represent $|A'(w_1,\ldots,w_l)\rangle$ as a superposition of the possible values of z, which completes the induction. ∎

Now suppose that P q–generates ψ (represented by M_ψ) with ϵ error and l communication qubits. Let us denote by $\phi = \sum_{x,y} a_{x,y}|x,y\rangle$ the final superposition that

the two parties compute (which is, again, represented by M_ϕ). By Claim 4.2 we know that we can represent ϕ as $\phi = \sum_{w\in\{0,1\}^l} |A(w), B(w)\rangle$. Because ϕ_{Alice} has support in X, and ϕ_{Bob} in Y, this is actually $\phi = \sum_{w\in\{0,1\}^l} \sum_{x,y} a_x(w)\cdot b_y(w)|x,y\rangle$, where $a_x(w)$ and $b_y(w)$ are complex numbers. Thus

$$M_\phi[x,y] = \sum_{w\in\{0,1\}^l} a_x(w)b_y(w).$$

Let us define an $|X|\times 2^l$ matrix A by $A[x,w] = a_x(w)$, and a $2^l\times|Y|$ matrix $B[w,y] = b_y(w)$. We see that $M_\phi = A\cdot B$, where the $\cdot$ operation is matrix multiplication. In particular,

$$\operatorname{rank}(M_\phi) \le \operatorname{rank}(A) \le 2^l.$$

Since ϕ ϵ-approximates ψ, we know that $||M_\phi - M_\psi||_2^2 = \langle M_\phi - M_\psi | M_\phi - M_\psi\rangle = 2(1-\langle\phi|\psi\rangle) \le 2\epsilon$. It follows that $K_{2\epsilon} = \min_{M:||M-M_\psi||_2^2\le 2\epsilon} \operatorname{rank}(M) \le \operatorname{rank}(M_\phi) \le 2^l$, as desired.

5. Relationships between sampling and computing. We say that a function $g: X\times Y \mapsto M$ is a "product" function if $g(x,y) = g_1(x)g_2(y)$ for some functions g_1 and g_2. For product distributions μ we show that sampling is not harder than q–generating, which in turn is not harder than worst-case solving the problem.

5.1. Sampling vs. q–generating.

LEMMA 5.1. *Suppose that $f: X\times Y\mapsto\{0,1\}$, and μ is an l_2* product *distribution. Let $\mathcal{D}: X\times Y\mapsto[0,1]$ be the classical distribution associated with μ, $\mathcal{D}(x,y) = |\mu_{x,y}|^2$. Then $\overset{\circ}{Q}_{4\sqrt{\epsilon}}(f,\mathcal{D}) \le \overset{\bullet}{Q}_\epsilon(f,\mu) + O(1)$.*

Proof. Suppose that the approximation protocol computes ϕ s.t. $|\langle\phi|\psi\rangle| \ge 1-\epsilon$, where ψ is the ideal superposition $\psi = \sum_{x,y}\mu_{x,y}(-1)^{f(x,y)}|x,y\rangle$. We give a sampling protocol:

1. Alice computes the superposition $|00\rangle + |11\rangle$ in qubits z_1, z_2. She sends the second qubit z_2 to Bob.
2. If they both have a $|0\rangle$ (i.e., $z_1 = z_2 = 0$), they compute in the qubits X, Y the superposition $\sum_{x,y}\mu_{x,y}|x,y\rangle$ (this can be done at no cost, as μ is a product distribution), and if they have a 1, they compute ϕ (using $\lceil\log(K_\epsilon)\rceil$ qubits).
3. Now Bob returns the qubit z_2 to Alice. Alice does a unitary transformation over z_1, z_2 that sends $|00\rangle$ to $\frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)$ and $|11\rangle$ to $\frac{1}{\sqrt{2}}(|00\rangle - |01\rangle)$.
4. Finally, both players measure all their qubits.

Now, suppose for the moment that the protocol was run with $\phi = \psi$. In that case after step 2 the two players are in the superposition

$$\sum_{x,y}\mu_{x,y}[|00,x,y\rangle + (-1)^{f(x,y)}|11,x,y\rangle].$$

It can then be easily verified that after step 3 the resulting superposition is

$$\sum_{x,y}\mu_{x,y}|0,f(x,y),x,y\rangle,$$

and thus when Alice and Bob measure their qubits, they actually sample f according to $\mathcal{D}$ with no error.

Now, in the actual protocol the two players compute ϕ, which is not quite ψ but close to it, namely, $|\langle\phi|\psi\rangle| \geq 1-\epsilon$. By Fact 3.1 we know that the resulting distribution is $2\sqrt{1-|\langle\phi|\psi\rangle|^2} \leq 2\sqrt{1-(1-\epsilon)^2} \leq 2\sqrt{2\epsilon}$ close (in the l_1 norm) to the right one, and the lemma follows. □

5.2. q–generating vs. computing. Suppose that we can compute f, and that we want to q–generate it according to a product distribution μ. Since μ is product, we can enter the superposition $\sum_{x,y} \mu_{x,y}|x,y\rangle$. Then we can compute f. However, this does not give a q–generating protocol because we might use some auxiliary qubits for the computation and thus have garbage entangled with the result. The following proof follows ideas from Cleve et al. [7], who showed how to remove such garbage. The proof is given here for completeness.

LEMMA 5.2. *For any function f and any l_2 product distribution μ, $\overset{\bullet}{Q}_{2\epsilon}(f,\mu) \leq 2Q_\epsilon(f)$.*

Proof. Let T be a small error protocol for computing f. We use the safe storage principle, and each time the protocol wants to measure a qubit we simply copy it to a new qubit that is left untouched. Now, say that $T|x,0,y,0\rangle = |x,y\rangle \otimes [\alpha^0_{x,y}|f(x,y),g^0_{x,y}\rangle + \alpha^1_{x,y}|1-f(x,y),g^1_{x,y}\rangle]$, where $g^0_{x,y}$ (and $g^1_{x,y}$) is the correlated garbage that is produced during the computation and is divided between the two players; i.e., the right answer $f(x,y)$ is computed with amplitude $\alpha^0_{x,y}$ and is accompanied by $g^0_{x,y}$ in the garbage qubits.

The two players get into the superposition $\sum_{x,y} \mu_{x,y}|x,y\rangle$. Since μ is a product distribution, this is done at no cost. We run the following three-step protocol:

Compute f. This results in

$$\phi_1 = \sum_{x,y} \mu_{x,y}\ |x,y\rangle \otimes [\alpha^0_{x,y}|f(x,y),g^0_{x,y}\rangle + \alpha^1_{x,y}|1-f(x,y),g^1_{x,y}\rangle].$$

As T has only ϵ error on average, we know that $\sum_{x,y}\ |\mu_{x,y}|^2\ |\alpha^0_{x,y}|^2 \geq 1-\epsilon$.

Lift the result. Next, we lift the result $f(x,y)$ to the amplitude; i.e., the player with the result qubit R changes the amplitude by $(-1)^R$. The resulting superposition is

$$\phi_2 = \sum_{x,y} \mu_{x,y}(-1)^{f(x,y)}|x,y\rangle \otimes [\alpha^0_{x,y}|f(x,y),g^0_{x,y}\rangle - \alpha^1_{x,y}|1-f(x,y),g^1_{x,y}\rangle].$$

Notice the sign change in the garbage belonging to the wrong answer. We do not like this sign change, and we notice that this sign change is immaterial. Namely, if we define

$$\psi_2 = \sum_{x,y} \mu_{x,y}(-1)^{f(x,y)}|x,y\rangle \otimes [\alpha^0_{x,y}|f(x,y),g^0_{x,y}\rangle + \alpha^1_{x,y}|1-f(x,y),g^1_{x,y}\rangle],$$

then $|\langle\phi_2|\psi_2\rangle| \geq \sum_{x,y} |\mu_{x,y}|^2(|\alpha^0_{x,y}|^2 - |\alpha^1_{x,y}|^2)$, which is at least $1-2\epsilon$.

Reverse the computation. Finally, we would like to get rid of the garbage, and so we reverse T; this at most doubles the number of communication qubits transferred. Because of the sign change in ϕ_2, the resulting superposition is ugly and depends on the actual computation. However, had the reversing step been applied to ψ_2, we would have received the ideal superposition $\psi = \sum_{x,y} \mu_{x,y}(-1)^{f(x,y)}|x,y\rangle$. Now $|\langle\phi_2|\psi_2\rangle| \geq 1-2\epsilon$, and reversing T is just a unitary transformation. We conclude that $|\langle\phi_3|\psi\rangle| \geq 1-2\epsilon$. □

6. The $DISJ_k$ function. The $DISJ_k(x,y)$ function gets as input two subsets $S,T \subseteq \{1,\ldots,n\}$, each of cardinality k, and outputs 1 iff $S \cap T = \emptyset$. We bound the quantum sampling complexity of the function under the l_2 uniform distribution $\mu_{x,y} = 1/N$. We then prove the following result.

THEOREM 6.1. *For $k = \Theta(\sqrt{n})$, $\overset{\circ}{Q}_\epsilon(DISJ_k) = O(\log(n)\log(\frac{1}{\epsilon}))$. The result is true even when Alice has an input S and Bob wants to sample a subset T disjoint from S.*

By Theorem 4.1 we need to analyze the spectrum of $M = M_{DISJ_k,\mu}$. Indeed, notice that $M[x,y]$ depends only on the intersection size of x and y. It is not too difficult to see that all matrices that are indexed by k-subsets and depend only on the intersection size commute. In particular, they share the same eigenspaces. Lovasz [13] analyzed the spectrum of these matrices, and we give a slightly different description of the eigenspaces of M than he obtains.

LEMMA 6.2 (a different presentation of [13]). *M has $k+1$ eigenspaces $E_0,\ldots,E_k$. E_0 is of dimension 1 and contains the all 1's vector. E_i has dimension $\binom{n}{i} - \binom{n}{i-1}$. The typical eigenvector in E_i is indexed by $x_1,x_2,\ldots,x_{2i-1},x_{2i} \in \{1,\ldots,n\}$. The corresponding eigenvector e (unnormalized) is given by $e_S = 0$ if there is an index $j : |S \cap \{x_{2j-1},x_{2j}\}| \neq 1$, and otherwise by $e_S = \Pi_j(-1)^{|S\cap\{x_{2j}\}|}$. The corresponding eigenvalues are*

$$\lambda_0 = \frac{2\binom{n-k}{k} - \binom{n}{k}}{\binom{n}{k}} \quad \textit{and} \quad \lambda_i = \frac{2\binom{n-k-i}{k-i}}{\binom{n}{k}}$$

for $i > 0$.

The eigenvalues in the spectrum of M decay rapidly. Let $q_i = \sum_{w_i \in E_i} |\lambda_i|^2$ so that $\sum_{i=0}^k q_i = 1$. Then the following holds.

CLAIM 6.1. *For $k = \Theta(\sqrt{n})$, $\frac{q_{i+1}}{q_i} = O(\frac{1}{i+1})$.*

Proof. To calculate q_{i+1}/q_i, we first bound λ_{i+1}/λ_i. We get that $\frac{-\lambda_{i+1}}{\lambda_i} = \frac{k-i}{n-k-i} \leq \frac{2k}{n}$. The number of eigenvalues is $\binom{n}{i} - \binom{n}{i-1}$ for E_i and $\binom{n}{i+1} - \binom{n}{i}$ for E_{i+1}, and

$$\frac{\binom{n}{i+1} - \binom{n}{i}}{\binom{n}{i} - \binom{n}{i-1}} \leq \frac{2n}{i+1}.$$

Hence

$$\frac{q_{i+1}}{q_i} = \frac{(\binom{n}{i+1} - \binom{n}{i})\lambda_{i+1}^2}{(\binom{n}{i} - \binom{n}{i-1})\lambda_i^2} \leq \frac{2n}{i+1} \cdot \frac{4k^2}{n^2} = \Omega\left(\frac{1}{i+1}\right). \qquad \square$$

Therefore $q_t \leq \frac{c^t}{t!}$. Now we are set to prove the following.

LEMMA 6.3. $\log K_\epsilon \leq O(\log(n)\frac{\log 1/\epsilon}{\log\log 1/\epsilon})$.

Proof. We set $t = O(\frac{\log 1/\epsilon}{\log\log 1/\epsilon})$ and take $\Lambda = E_0 \cup E_1 \cup \cdots \cup E_t$. We have

$$\sum_{i\in\Lambda} |\lambda_i|^2 = 1 - \sum_{i\notin\Lambda} |\lambda_i|^2 \;=\; 1 - \sum_{i=t+1}^{k} q_i$$
$$\geq 1 - \sum_{i=t+1}^{k} \frac{c^i}{i!} \geq 1 - O\left(\frac{c^t}{t!}\right) \geq 1 - \epsilon.$$

Hence $K_\epsilon \le |E_0 \cup \cdots \cup E_t| \le t \cdot \binom{n}{t} \le n^{t+1}$, and $\log K_\epsilon \le (t+1)\log(n)$, as required. □

By Theorem 4.1, $\overset{\bullet}{Q}_\epsilon(\psi) \le \lceil \log K_\epsilon \rceil \le O(t\log(n)) = O(\log(n)\log 1/\epsilon)$, and a similar upper bound on $\overset{\circ}{Q}_\epsilon$ $(DISJ_k)$ follows from Lemma 5.1. This gives the first part of Theorem 6.1. This, in particular, shows that it is easy for Alice and Bob to sample a uniform pair of subsets x and y, along with the knowledge as to whether x and y intersect.

In the next two subsections we prove the second part of the theorem. We want to show two things. One is that the result holds even when Alice and Bob want to sample only *disjoint* subsets, and second that the result holds even when Alice is given an input x and Bob is asked to sample a subset y *disjoint* with x.

6.1. Generating disjoint subsets. Alice and Bob want to ϵ-approximate a sample of disjoint k-subsets x and y. This amounts to sampling the disjointness function according to the distribution $\mathcal{D}$ that is uniform over all pairs of disjoint subsets. (Notice that $\mathcal{D}$ is *not* a product distribution.) Clearly, it is enough for Alice and Bob to approximate the normalized superposition $\psi = \sum_{x,y:x\cap y=\emptyset} \frac{1}{\sqrt{\Delta_0 N}}|x,y\rangle$, for once they do that they can measure x and y and get the desired sample. The normalizing factor Δ_0 is the number of values y in a row x s.t. $x \cap y = \emptyset$ and does not depend on x.

Denote by M_{f_0} the normalized matrix

$$M_{f_0}[x,y] = \frac{1}{\sqrt{\Delta_0 N}} \begin{cases} 1, & x \cap y = \emptyset, \\ 0, & \text{otherwise.} \end{cases}$$

M_{f_0} is symmetric and has full spectrum $\zeta_1, \ldots, \zeta_N$, $|\zeta_1|^2 \ge \cdots |\zeta_N|^2$. We say K^0_ϵ is the first K s.t. $\sum_{i\le K} |\zeta_i|^2 \ge 1-\epsilon$. By Theorem 4.1 (which applies to any superposition), Alice and Bob can ϵ-approximate ψ using only $O(\log(K^0_\epsilon))$ communication qubits.

Since $k = \Theta(\sqrt{N})$, $\Delta_0 \ge \frac{N}{c}$ for some constant c. It is left to show that $O(\log(K^0_\epsilon)) = O(\log(n)\log(1/\epsilon))$. One way to show this is to compute the eigenvalues of M_{f_0}. However, there is an easier way. We show that $K^0_\epsilon \le K_{\epsilon/c} + 1$ and then Lemma 6.3 implies the bound. We are left with the following.

CLAIM 6.2. $K^0_\epsilon \le K_{\epsilon/c} + 1$.

Proof. Denote $M_f[x,y] = \frac{1}{N}(-1^{f(x,y)}))$. M_f and M_{f_0} share the same eigenspaces (as they commute). We now express M_f and M_{f_0} in terms of each other. Let us define $B = \sqrt{N\Delta_0}M_{f_0}$, so that B is a 0,1 matrix. It can easily be verified that

$$NM_f = B - (J - B) = 2B - J,$$

where J is the all 1's matrix. Hence, $M_{f_0} = N/2\sqrt{N\Delta_0}M_f + dJ$ for some value d. In particular, for any eigenvector $w_i \ne (1,\ldots,1)$, $Jw_i = 0$ and $\zeta_i = \frac{1}{2}\sqrt{N/\Delta_0}\lambda_i$. Thus,

$$|\zeta_i| = \frac{1}{2}\sqrt{\frac{N}{\Delta_0}}|\lambda_i| \le \sqrt{c}|\lambda_i|, \quad i > 1.$$

Therefore, suppose $\sum_{i\in S}|\lambda_i|^2 \ge 1 - \epsilon/c$. Denote $S' = S \cup \{(1,\ldots,1)\}$. Clearly, $\sum_{i\notin S'}|\zeta_i|^2 \le \sum_{i\notin S'} c|\lambda_i|^2 \le \epsilon$. Hence $K^0_\epsilon \le K_{\epsilon/c} + 1$. □

6.2. Sampling for a given input x. Alice is given an input $z \in X$, and the goal is that Bob samples $y \in Y$ s.t. $z \cap y = \emptyset$. We follow a protocol similar to that in the upper bound of Theorem 4.1. Given an input $z \in X$ and an $\epsilon > 0$, define

$M = M_{f_0}$ as in the previous subsection. Let W be the eigenvector basis of M (which is symmetric). Let $\Lambda = \Lambda_\epsilon$ be the union of the first eigenspaces E_i (defined in Lemma 6.2) that contain the first K_ϵ heavy eigenvectors of M. Let Π be the projection operator over Λ.

We now describe the protocol. Alice gets into the normalized superposition $v_z = \frac{1}{\sqrt{\Delta_0}} \sum_{y:y\cap z=\emptyset} |y\rangle$. In the eigenvector basis W, $v_z = \sum_i \gamma_i |w_i\rangle$. Alice then projects v_z onto Λ to get $\overline{v_z} = \sum_{i\in\Lambda} \gamma_i |w_i\rangle$ and sends $\overline{v_z}$ to Bob. Bob returns $\overline{v_z}$ to the original basis and measures to get some y. To prove correctness we show the following.

LEMMA 6.4. $|\langle v_z|\overline{v_z}\rangle| \geq 1 - \epsilon$.

Proof. $\langle v_z|\overline{v_z}\rangle = \sum_{i\in\Lambda} |\gamma_i|^2$, i.e., it is the length of the projection of v_z onto Λ. We show that this quantity is the same for all z. If we know that, we can define $\psi = \frac{1}{\sqrt{N}} \sum_z |z, v_z\rangle$ and $\overline{\psi} = \frac{1}{\sqrt{N}} \sum_z |z, \overline{v_z}\rangle$ (so ψ and $\overline{\psi}$ are normalized). Then, from the proof of Theorem 4.1 we know that

$$|\langle \psi|\overline{\psi}\rangle| \geq 1 - \epsilon.$$

However,

$$\langle \psi|\overline{\psi}\rangle = \frac{1}{N} \sum_z \langle v_z|\overline{v_z}\rangle = \langle v_z|\overline{v_z}\rangle,$$

which together implies that $\langle v_z|\overline{v_z}\rangle = \langle \psi|\overline{\psi}\rangle \geq 1-\epsilon$, as required. Indeed, the following claim holds.

CLAIM 6.3. *For any eigenspace E_j, $|\langle v_z|E_j\rangle|^2$, which is the length of the projection of v_z on E_j, does not depend on z.*

Proof. Let $z_1, z_2 \in X$ be two k-subsets; i.e., $z_1, z_2 \subset [1, \ldots, n]$ and $|z_1| = |z_2| = k$. There is a permutation $\pi \in S_n$ s.t. $\pi(z_1) = z_2$, where for a set A, $\pi(A) = \{\pi(a)|a \in A\}$.

The operation of the permutation π can be thought of as a unitary transformation permuting the basis vectors $|x\rangle$ for $x \in X$. In other words, given a superposition $\phi = \sum_{i\in X} a_i|i\rangle$, $\pi(\phi)$ is defined to be $\sum_{i\in X} a_i|\pi(i)\rangle$. In particular, for any two superpositions ϕ_1, ϕ_2, $\langle \pi(\phi_1)|\pi(\phi_2)\rangle = \langle \phi_1|\phi_2\rangle$. As a result, $\langle v_{z_1}|E_j\rangle = \langle \pi(v_{z_1})|\pi(E_j)\rangle$, where $\pi(E_j) = Span\{\pi(w)|w \in E_j\}$. However, we observe that

$$\begin{aligned}
\pi(v_{z_1}) &= \sum_{y:y\cap z_1=\emptyset} |\pi(y)\rangle \\
&= \sum_{w:\pi^{-1}(w)\cap z_1=\emptyset} |w\rangle \\
&= \sum_{w:w\cap\pi(z_1)=\emptyset} |w\rangle \;=\; v_{z_2}.
\end{aligned}$$

Finally, because of the symmetries of the eigenspaces E_j, $\pi(E_j) = E_j$. The lemma follows. □ □

7. A lower bound on classical sampling. In contrast we prove that classically sampling $DISJ_k$ is hard. We begin with the observation that classical sampling protocols can always be made to have just one message at no cost. We then prove the following result.

LEMMA 7.1. *Given any sampling protocol P with k communication bits and ϵ error, there is an optimal one message sampling protocol that samples from the desired distribution with the same complexity.*

Proof. The protocol goes as follows:

- Alice simulates the protocol P, playing the role of both players. She then announces the resulting sequence of messages M to Bob.[1]
- Alice and Bob pick inputs S and T according to the respective conditional distributions for the protocol P given the messages M.

The crucial observation is that, conditioned on the sequence of messages exchanged, the distribution from which Alice and Bob sample is a product distribution. □

We are now ready to prove the next result.

THEOREM 7.2. *Let $k = \sqrt{n}$. There is a constant $\epsilon > 0$ s.t. $\overset{\circ}{R}_\epsilon(DISJ_k) = \Omega(\sqrt{n})$.*

Proof. Let P be the distribution on $X \times Y$ that Alice and Bob sample from (X and Y is the set of all k-sets). By Lemma 7.1, P is a convex combination of L *product* distributions D_M, $P = \sum p_i D_i$, where L is the size of the message space from which Alice chooses her message to Bob (i.e., $\log L$ is the number of bits transmitted during the protocol). We say that a distribution D on rectangle R is *smooth* if for any pair of elements $u, v \in R$, $\frac{D(u)}{D(v)} \leq 4$. We soon show that any product distribution can be very closely approximated by a convex combination of a small number of smooth distributions on rectangles; namely, we have the following.

CLAIM 7.1. *Let D be a product distribution on $X \times Y$. Then there are rectangles $R_1, \ldots, R_{9n^2}$, and smooth distributions D_i on R_i, such that D is within (total variation distance) 2^{-2n+1} of a convex combination of D_i.*

In particular, P is 2^{-n+1} close to a convex combination $\sum_{i=1}^{9n^2 L} p_i P_i$, where P_i is some smooth distribution over some rectangle R_i. Intuitively, the proof shows that large rectangles R_i introduce large error, while small rectangles provide very slow progress. For that we use the following combinatorial lemma of Babai, Frankl, and Simon.

LEMMA 7.3 (see [4]). *There exist a constant $\epsilon_0 > 0$ and $\delta = 2^{-\Omega(\sqrt{n})}$ such that, for any rectangle $R = U \times V$ with $\frac{|R|}{|X||Y|} \geq \delta$, at least ϵ_0 fraction of the pairs of subsets in R intersect (are not disjoint).*

Let us call a rectangle R_i large if $\frac{|R_i|}{|X||Y|} \geq \delta$. By the lemma, any large rectangle must contain at least ϵ_0 fraction of intersecting pairs. Thus, any smooth distribution P_i on a large rectangle R_i must have at least $\frac{\epsilon_0}{4}$ weight on intersecting pairs. Let h denote the total weight of heavy rectangles in the convex combination (i.e., $h = \sum_{i:R_i \text{ is heavy}} p_i$). We see that intersecting pairs get at least weight $\frac{h\epsilon_0}{4}$. We conclude that $\frac{h\epsilon_0}{4} \leq \epsilon$ and $h \leq \frac{4\epsilon}{\epsilon_0} = O(\epsilon)$.

We now concentrate on the nonheavy rectangles P_i. We say we *touch* a pair (x, y) if some nonheavy rectangle R_i contains it. Let I be the set of all disjoint pairs. We see that we must touch at least $(1 - (\epsilon + h))|I|$ pairs in I, or else there are $(\epsilon + h)|I|$ elements that get weight $\epsilon + h$ in the uniform distribution over disjoint pairs and only weight h in P. As every nonheavy rectangle R_i can touch at most $|R_i| \leq \delta|X| \cdot |Y|$ elements, we must have that $9n^2 L\delta|X| \cdot |Y| \geq (1 - \epsilon - h)|I| \geq (1 - O(\epsilon))|I|$.

For $k = \sqrt{n}$ the number of disjoint pairs is some $c_0|X| \cdot |Y|$ for some constant c_0. Thus, $L \geq (1 - O(\epsilon))c_0/9n^2 \cdot 2^{\Omega(\sqrt{n})}$. It follows that for some small enough constant $\epsilon > 0$ we must have $L \geq 2^{\Omega(\sqrt{n})}$, as desired.

We are left with the proof of Claim 7.1, which we give now.

[1] We assume that all messages belonging to the same round have the same length. If this is not the case, Alice has to send a special sign at the end of each message, which may, at most, increase the number of communication bits by a constant factor.

Proof of Claim 7.1. We partition X to sets $X_0, \ldots, X_{3n-1}$ and X_{Bad}, where $X_i = \{x | \frac{1}{2^{i+1}} \leq D(x) \leq \frac{1}{2^i}\}$ and X_{Bad} is all other strings. We similarly partition Y. We define the distribution $D_{i,j}$ to be the distribution that D induces on the rectangle $X_i \times Y_j$ $(0 \leq i, j \leq 3n-1)$. It is clear that $D_{i,j}$ is almost uniform. Let us denote by $\overline{D}$ the appropriate linear combination of the distributions $D_{i,j}$, $\overline{D} = \sum_{i,j} p_{i,j} D_{i,j}$ (where $p_{i,j}$ is the weight of the rectangle $X_i \times Y_j$ under D). It is clear that $\overline{D}(a,b) = D(a,b)$ for any (a,b) that belongs to some rectangle $X_i \times X_j$. Thus, $|\overline{D} - D|_1$ is bounded by the total weight (under D) of entries in $X_{Bad} \times Y$ and $X \times Y_{Bad}$, and so it is bounded by $2 \cdot 2^n \cdot 2^{-3n} = 2^{-2n+1}$. The lemma follows. □ □

8. Zero error sampling and the log-rank conjecture. Theorem 4.1 characterizes the q–generating complexity $\overset{\bullet}{Q}$. However, it is still possible that sampling is much easier (even in the quantum world) than q–generating. For the special case of *zero error* sampling, we supply a lower bound even for the easier task of sampling, using a method similar to that used in Theorem 4.1.

THEOREM 8.1. *For every function f and any distribution $\mathcal{D}$, $\overset{\circ}{Q}_0(f, \mathcal{D}) \geq \frac{\log(\mathrm{rank}(M_{f,\mathcal{D}}))}{2} - 1$.*

Proof. Given a protocol P for sampling f, we define the $|X| \times |Y|$ matrix M_P^0 by letting $M_P^0[x,y]$ be the probability that P samples (x,y) with the answer 0. We similarly define M_P^1. We let $M_P = M_P^0 - M_P^1$. Note that M_P does not necessarily correspond to the probability that the protocol will answer with a yes or no to an instance (x,y).

LEMMA 8.2 (see [12]). *Suppose that P uses only l communication qubits. Then* $\mathrm{rank}(M_P^0), \mathrm{rank}(M_P^1) \leq 2^{2l}$.

Proof. Let P be a quantum protocol for sampling f using l qubits. Suppose by the end of the protocol that the superposition is ϕ, and w_l, the last qubit communicated, contains the answer (0 or 1). By Claim 4.2,

$$\phi = \sum_{w \in \{0,1\}^l} \sum_{x \in X, y \in Y} |x, U_x(w), w_l, y, V_y(w)\rangle.$$

Define $Y_0 = \{w \in \{0,1\}^l | w_l = 0\}$ and $\phi_{x,y}^0 = \sum_{w \in Y_0} |x, U_x(w), w_l, y, V_y(w)\rangle$. Then

$$\begin{aligned} M_P^0[x,y] &= \langle \phi_{x,y}^0 | \phi_{x,y}^0 \rangle \\ &= \sum_{w,z \in Y_0} \langle U_x(w) | U_x(z) \rangle \langle V_y(w) | V_y(z) \rangle. \end{aligned}$$

If we define a matrix A of dimension $|X| \times |Y_0|^2$ by $A[x,(w,z)] = \langle U_x(w) | U_x(z) \rangle$, and a matrix B of dimension $|Y_0|^2 \times |Y|$ by $B[(w,z),y] = \langle V_y(w) | V_y(z) \rangle$, then we see that $M_P^0[x,y] = (AB)[x,y]$. That is, $M_P^0 = AB$. In particular, $\mathrm{rank}(M_P^0) = \mathrm{rank}(AB) \leq \mathrm{rank}(A) \leq |Y_0|^2 \leq 2^{2l}$. A similar argument shows that $\mathrm{rank}(M_P^1) \leq 2^{2l}$. □

We remind the reader that for $f : X \times Y \mapsto \{0,1\}$ the matrix $M_{f,\mathcal{D}}$ has dimensions $|X| \times |Y|$ and is defined by $M_{f,\mathcal{D}}[x,y] = (-1)^{f(x,y)} \mathcal{D}_{x,y}$. ($M_{f,\mathcal{D}}$ is not normalized, i.e., $||M_{f,\mathcal{D}}||_2$ is not necessarily 1.) We notice that if P samples f with zero error using l qubits, then $M_P = M_{f,\mathcal{D}}$. Moreover, $\mathrm{rank}(M_{f,\mathcal{D}}) = \mathrm{rank}(M_P) \leq \mathrm{rank}(M_P^0) + \mathrm{rank}(M_P^1) \leq 2^{2l} + 2^{2l} = 2^{2l+1}$. In particular, $2l + 1 \geq \log(\mathrm{rank}(M_{f,\mathcal{D}}))$. Hence $\overset{\circ}{Q}_0(f, \mathcal{D}) \geq \log(\mathrm{rank}(M_{f,\mathcal{D}}))/2 - 1$, and Theorem 8.1 follows. □

We see in particular that for the uniform distribution ($\mathcal{D}(x,y) = 1/N^2$ and $\mu(x,y) = 1/N$), $M_{f,\mathcal{D}}$ and $M_{f,\mu}$ differ only by a constant factor and so have the same

rank. By Theorem 4.1, $\overset{\circ}{Q}_0(f) \leq \lceil \log \operatorname{rank}(M_f M_f^\dagger) \rceil + O(1) = \lceil \log \operatorname{rank}(M_f) \rceil + O(1)$. Theorem 8.1 gives a matching lower bound. Together we get the following tight characterization for zero error sampling.

COROLLARY 8.3. *For any* $f : X \times Y \mapsto \{0,1\}$, $\overset{\circ}{Q}_0(f) = \Theta(\log \operatorname{rank}(M_f))$.

8.1. Zero error classical computing is almost as easy as sampling.

THEOREM 8.4. $\sqrt{D(f)} \leq \overset{\circ}{R}_0(f) \leq D(f)$.

Proof. Given the matrix M_f, a monochromatic cover is a set of monochromatic rectangles in M_f that together cover the whole matrix. Define $C(f)$ as the smallest number of monochromatic rectangles needed to cover M_f. Define $C^D(f)$ as the smallest number of disjoint monochromatic rectangles needed to cover M_f. It is well known (see [16, Chapter 2]) that

$$\sqrt{D(f)} \leq N(f) = \log_2 C(f) \leq \log_2 C^D(f) \leq D(f),$$

where $N(f)$ is the nondeterministic communication complexity. We show that

$$\log_2 C(f) \leq \overset{\circ}{R}_0(f) \leq \log_2 C^D(f),$$

and in particular we get that $\sqrt{D(f)} \leq N(f) \leq \overset{\circ}{R}_0(f) \leq D(f)$, as required.

We first show that $\log_2 C(f) \leq \overset{\circ}{R}_0(f)$. Indeed, by Lemma 7.1 there is a one message zero error sampling protocol whose complexity is $k = \overset{\circ}{R}_0(f)$. In the one message protocol a message M is chosen (out of the 2^k possible messages) according to some probability distribution, and, given the message M, Alice (Bob) chooses a message $x \in X$ ($y \in Y$) according to some probability distribution that depends on M. Let us say that X_M (Y_M) is the set of elements in X that have nonzero probability of being selected by Alice (Bob), given the message M. As the protocol has zero error, the rectangle $X_M \times Y_M$ must be monochromatic. As Alice and Bob sample inputs according to the uniform distribution, every $(x,y) \in X \times Y$ must be covered. Hence the protocol gives rise to a monochromatic cover of M_f with only 2^k rectangles, and hence $C(f) \leq 2^k$.

Next we show that $\overset{\circ}{R}_0(f) \leq \log_2 C^D(f)$. Suppose that a disjoint monochromatic cover of M_f with 2^k rectangles exists. Say that the cover contains the rectangles $R_1, \ldots, R_{2^k}$ and $R_i = X_i \times Y_i$. We build a sampling protocol. A message $i \in \{1, \ldots, 2^k\}$ is picked with probability proportional to the area of R_i. Given the message i, Alice picks a random element $x \in X_i$, and Bob picks a random element $y \in Y_i$. It is easy to verify that, as the cover is disjoint, this results in the uniform distribution over $X \times Y$ along with the value of $f(x,y)$. Hence $\overset{\circ}{R}_0(f) \leq k$. ∎

Acknowledgments. We thank Dorit Aharonov, Ike Chuang, Michael Nielsen, and Steven Rudich for very helpful discussions. We also thank the anonymous referees for many helpful comments.

REFERENCES

[1] S. AARONSON AND A. AMBAINIS, *Quantum Search of Spatial Regions*, Technical report, in quant-ph/0303041.

[2] D. AHARONOV, A. KITAEV, AND N. NISAN, *Quantum circuits with mixed states*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), Dallas, TX, 1998, ACM, New York, pp. 20–30.

[3] A. Ambainis, A. Nayak, A. Ta-Shma, and U. Vazirani, *Dense quantum coding and quantum finite automata*, J. ACM, 49 (2002), pp. 496–511.

[4] L. Babai, P. Frankl, and J. Simon, *Complexity classes in communication complexity theory*, in Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS), 1986, Toronto, IEEE Computer Society Press, Washington, DC, 1986, pp. 337–347.

[5] C. Bennett and S. Wiesner, *Communication via one- and two- particle operators on Einstein–Podolsky–Rosen states*, Phys. Rev. Lett., 69 (1992), pp. 2881–2884.

[6] H. Buhrman, R. Cleve, and A. Wigderson, *Quantum vs. classical communication and computation*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), Dallas, TX, 1998, ACM, New York, 1998, pp. 63–68.

[7] R. Cleve, W. van Dam, M. Nielsen, and A. Tapp, *Quantum entanglement and the communication complexity of the inner product function*, Lecture Notes in Comput. Sci., 1509 (2002), pp. 61–74.

[8] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, Proc. Roy. Soc. London Ser. A, 439 (1992), pp. 553–558.

[9] L. K. Grover, *A fast quantum mechanical algorithm for database search*, in Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC), Philadelphia, PA, 1996, ACM, New York, 1996, pp. 212–219.

[10] A. Holevo, *Bounds for the quantity of information transmitted by a quantum communication channel*, in Problemy Peredachi Informatsii, 9 (1973), pp. 3–11; English translation Prob. Inf. Transm., 9 (1973), pp. 177–183.

[11] R. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1987.

[12] I. Kremer, *Quantum Communication*, Master's thesis, The Hebrew University of Jerusalem, Jerusalem, 1995.

[13] L. Lovasz, *On the Shannon capacity of a graph*, IEEE Trans. Inform. Theory, 25 (1979), pp. 1–7.

[14] A. Nayak, *Optimal lower bounds for quantum automata and random access codes*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS), New York, 1999, IEEE Computer Society Press, Washington, DC, 1999, pp. 369–376.

[15] N. Nisan and A. Wigderson, *On rank vs. communication complexity*, Combinatorica, 15 (1995), pp. 557–566.

[16] N. Nisan and E. Kushilevitz, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.

[17] A. A. Razborov, *Quantum communication complexity of symmetric predicates*, Izvestiya Math, 67 (2003) (in Russian); English version at quant-ph/0204025.

[18] R. Raz and B. Spieker, *On the "log rank"-conjecture in communication complexity*, Combinatorica, 15 (1995), pp. 567–588.

[19] R. Raz, *Exponential separation of quantum and classical communication complexity*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), Atlanta, GA, 1999, ACM, New York, 1999, pp. 358–367.

[20] A. C. Yao, *Quantum circuit complexity*, in Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS), Palo Alto, CA, 1993, IEEE Computer Society Press, Silver Springs, MD, 1993, pp. 352–361.

[21] A. C. Yao, *Some complexity questions related to distributive computing*, in Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC), Atlanta, GA, 1979, ACM, New York, pp. 209–213.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1586–1620

THE EXPECTED NUMBER OF 3D VISIBILITY EVENTS IS LINEAR*

OLIVIER DEVILLERS†, VIDA DUJMOVIĆ‡, HAZEL EVERETT§, XAVIER GOAOC§, SYLVAIN LAZARD§, HYEON-SUK NA¶, AND SYLVAIN PETITJEAN§

Abstract. In this paper, we show that, amongst n uniformly distributed unit balls in $\mathbb{R}^3$, the expected number of maximal nonoccluded line segments tangent to four balls is linear. Using our techniques we show a linear bound on the expected size of the visibility complex, a data structure encoding the visibility information of a scene, providing evidence that the storage requirement for this data structure is not necessarily prohibitive. These results significantly improve the best previously known bounds of $O(n^{8/3})$ [F. Durand, G. Drettakis, and C. Puech, *ACM Transactions on Graphics*, 21 (2002), pp. 176–206].

Our results generalize in various directions. We show that the linear bound on the expected number of maximal nonoccluded line segments that are not too close to the boundary of the scene and tangent to four unit balls extends to balls of various but bounded radii, to polyhedra of bounded aspect ratio, and even to nonfat three-dimensional objects such as polygons of bounded aspect ratio. We also prove that our results extend to other distributions such as the Poisson distribution. Finally, we indicate how our probabilistic analysis provides new insight on the expected size of other global visibility data structures, notably the aspect graph.

Key words. computational geometry, three-dimensional visibility, visual events, visibility complex, probabilistic analysis, expected complexity

AMS subject classifications. 68U05, 60D05

DOI. 10.1137/S0097539702419662

1. Introduction.

Visibility computations are central in computer graphics applications. Computing the limits of the umbra and penumbra cast by an area light source, identifying the set of blockers between any two polygons, and determining the view from a given point are examples of visibility queries that are essential for the realistic rendering of three-dimensional (3D) scenes. In global illumination algorithms, where the flow of light in a scene is simulated according to the laws of geometrical optics, visibility computations are excessively costly. In fact, more than half of the overall computation time can routinely be spent on visibility queries in radiosity simulations [11].

One approach to speeding up rendering is to store global visibility information in a data structure which can then be efficiently queried. The visibility complex, a partition of the set of maximal-free line segments, has been proposed as a unified data structure encoding the visibility information of a scene [21] and has been used for

*Received by the editors December 16, 2002; accepted for publication (in revised form) June 4, 2003; published electronically October 14, 2003.

http://www.siam.org/journals/sicomp/32-6/41966.html

†INRIA Sophia-Antipolis, BP 93, 06902 Sophia-Antipolis Cedex, France (Olivier.Devillers@inria.fr, http://www-sop.inria.fr/prisme/). The work of this author was partially supported by the IST Programme of the EU as a Shared-Cost RTD (FET Open) Project under contract IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces).

‡School of Computer Science, McGill University, Ottawa, ON, Canada (vida@cs.mcgill.ca). This author's research was supported by FCAR.

§LORIA - INRIA Lorraine, CNRS, Univ. Nancy 2, France (everett@loria.fr, http://www.loria.fr/∼everett/; goaoc@loria.fr, http://www.loria.fr/∼goaoc/; lazard@loria.fr, http://www.loria.fr/∼lazard/; petitjea@loria.fr, http://www.loria.fr/∼petitjea/). The research of these authors was supported by the McGill-ISA collaborative INRIA project.

¶School of Computing, Soongsil University, Seoul, South Korea (hsnaa@computing.ssu.ac.kr). This author's research was done during a postdoctoral tenure at LORIA - INRIA Lorraine.

rendering purposes [9]. Other related data structures include Pellegrini's ray-shooting structure [18], the aspect graph [20], and the visual hull [12]; see [7] for a recent survey.

One problem with these types of data structures which may prevent their application in practice is their potentially enormous size; the size of the visibility complex of a set of n triangles in $\mathbb{R}^3$ is $\Theta(n^4)$ in the worst case [9], which is prohibitive even for scenes of relatively modest size. Worst-case examples are somewhat artificial, and indeed Durand, Drettakis, and Puech [8] provide empirical evidence indicating that these worst-case upper bounds are largely pessimistic in practical situations; they observe a quadratic growth rate, albeit for rather small scenes. In two dimensions, while the worst-case complexity of the visibility complex is quadratic, experimental results strongly suggest that the size of the visibility complex of a scene consisting of scattered triangles is linear [4].

Our goal is to provide theoretical evidence to support these observations. To this end we investigate the *expected size* of the visibility complex or, equivalently, the expected number of visibility events occurring in scenes in $\mathbb{R}^3$. A visibility event is a combinatorial change in the view of a moving observer; such an event occurs when the viewing direction becomes tangent to some objects. For sets of convex objects in general position in $\mathbb{R}^3$, the viewing direction can be tangent to at most four objects. Visibility events thus correspond to maximal nonoccluded line segments tangent to at most four objects; combinatorially different visibility events correspond to the faces of the visibility complex.

In this paper we prove that the expected number of maximal nonoccluded line segments tangent to four balls, amongst n uniformly distributed unit balls in $\mathbb{R}^3$, is linear. This improves the previously known upper bound of $O(n^{8/3})$ by Durand et al., who proved the more general result that the expected number of (possibly occluded) lines tangent to four balls is $O(n^{8/3})$ for the same model [9]. The intuition behind our proof is that, given a line segment tangent to four balls, the probability that this segment is not occluded by any other ball is the probability that a cylinder-like volume of radius 1 about the segment is free from the centers of the other balls. This probability decays roughly exponentially fast with the length of the segment, yielding the result. Using our techniques we then show a linear bound on the expected size of the visibility complex of n uniformly distributed unit balls in $\mathbb{R}^3$. A simple computation then provides us with the same result for the Poisson distribution.

Our results generalize in the following ways. We show that, for certain types of visibility events, the linear bound also applies to balls of various but bounded radii, to polyhedral objects enclosed between two concentric balls of fixed radius, and even to nonfat objects such as polygons, enclosed between two concentric circles of fixed radius, whose centers and normals are uniformly distributed. For the remaining types of visibility events (namely those occurring close to the boundary of the scene; see section 7.3 for the details), we prove only an $O(n^2)$ bound, which is still an improvement over the bound by Durand, Drettakis, and Puech [9].

Of course, objects in graphics scenes are seldom distributed uniformly or according to a Poisson point process. We chose this model because it allows tractable proofs of theoretical results. This is important in a context where there are few rigorous results, either theoretical or experimental. The same model, albeit with significant simplifying assumptions, has also been used to study the average complexity of ray shooting [23, 24] and occlusion culling for two-dimensional urban scenes [16]. It is interesting to note that Szirmay-Kalos et al. [23], after establishing bounds on the average complexity of ray shooting in scenes consisting of unit balls distributed

TABLE 1.1

Known bounds on the complexity of the set of lines, free lines, or maximal-free line segments tangent to 4 amongst n objects. The expected complexities are calculated for the uniform distribution. The results referenced by ⋆ are established in this paper.

	Worst-case	Expected
Possibly occluded lines amongst unit balls	$\Theta(n^4)$	$O(n^{\frac{8}{3}})$ [9]
Free lines amongst unit balls	$\Omega(n^2)$ [⋆], $O(n^{3+\epsilon})$ [1]	$\Theta(n)$ [⋆]
Free lines amongst disjoint homothetic polytopes	$\Omega(n^3)$ [3]	?
Free segments amongst unit balls	$\Omega(n^2)$ [⋆], $O(n^4)$	$\Theta(n)$[⋆]
Free segments amongst arbitrary sized balls	$\Omega(n^3)$ [6], $O(n^4)$	?
Visibility complex of unit balls	$\Omega(n^2)$ [⋆], $O(n^4)$	$\Theta(n)$ [⋆]

according to a Poisson point process, tested their algorithms on a small number of realistic scenes. The results they obtain are consistent with those predicted by the theoretical results, thus providing some evidence that the model is helpful. No other model has been widely accepted by the graphics community, and, in fact, generating meaningful random scenes usable for testing algorithms is a major problem. (Note that rather than attempting to generate random scenes, an alternative approach, which has been used to study the average complexity of ray shooting, is to fix the scene and randomly distribute the rays; see, for example, [2].)

Previous results on this topic include those that bound the number of lines and the number of free (i.e., nonoccluded) lines amongst different sets of objects. They are summarized in Table 1.1. Agarwal, Aronov, and Sharir [1] showed an upper bound of $O(n^{3+\epsilon})$ on the complexity of the space of line transversals of n balls by studying the lower envelope of a set of functions. A study of the upper envelope of the same set of functions yields the same upper bound on the number of free lines tangent to four balls [6]. Agarwal, Aronov, and Sharir [1] also showed a lower bound on the complexity of the space of line transversals of n balls of $\Omega(n^3)$ for arbitrarily sized balls and $\Omega(n^2)$ for unit sized balls. De Berg, Everett, and Guibas [3] showed a $\Omega(n^3)$ lower bound on the number of free lines (and thus free segments) tangent to 4 amongst n disjoint homothetic convex polyhedra. Recently, Devillers and Ramos [6] presented a simple $\Omega(n^3)$ lower bound on the number of free segments tangent to 4 amongst n arbitrarily sized balls, which also holds for nonintersecting balls. We also present a simple $\Omega(n^2)$ lower bound on the number of free segments tangent to 4 amongst n unit balls.

In the next section we carefully define the problem and state our main results. In section 3 and section 4 we prove the expected upper and lower linear bounds on the number of free segments tangent to four balls. In section 5 we extend this result to the visibility complex. We present in section 6 a $\Omega(n^2)$ worst-case lower bound. In section 7 we discuss extensions of our results to some other models. We conclude in section 8.

2. Our model and results. We first describe our objects and their distribution. Let $n \in \mathbb{N}$ and μ be a positive constant. A sample scene consists of n unit radius balls $B_1, \ldots, B_n$ whose centers $p_1, \ldots, p_n$ are independently chosen from the uniform distribution over a universal ball $\mathcal{U}$ of radius R centered at O. Since we distribute the centers p_i over $\mathcal{U}$, the balls B_i may intersect each other and are contained in the ball, denoted $\mathcal{U}^+$, whose radius is $R+1$ and whose center is that of $\mathcal{U}$.

We define the radius R of the universal ball $\mathcal{U}$ to be a function of n satisfying

$$R^3 = n/\mu. \tag{2.1}$$

The constant μ reflects the density of the balls in the sense that the expected number of centers lying in any given solid of volume V in the universe is $\frac{3}{4\pi}\mu V$. (The model is interesting only if n is asymptotically proportional to R^3. Indeed, if $\frac{n}{R^3}$ tends to infinity when n tends to infinity, then the universe gets entirely filled up with balls, and visibility events occur only in $\mathcal{U}^+ \setminus \mathcal{U}$. Conversely, if $\frac{n}{R^3}$ tends to zero when n tends to infinity, then the balls get scattered so far apart that the probability that any four (or three) balls have a common tangent goes to zero.)

We now define the *visibility complex* of a set of objects [21]. A *free* or *nonoccluded* segment is a line segment that does not intersect the interior of any object. A free segment is maximal if it is not properly contained in another one. Thus, the endpoints of a maximal-free segment are either on an object or at infinity. We say that two maximal-free segments are similar if their endpoints lie on the same objects (possibly at infinity). The visibility complex of a collection of objects is roughly defined as the partition of the space of maximal-free segments into connected components of similar segments.[1] Its faces have dimension between 0 and 4; when the objects are in adequate general position, a k-dimensional face corresponds to a connected set of similar maximal nonoccluded line segments tangent to $4-k$ objects.

In order to bound the total number of faces of the visibility complex, we first bound the number of 0-faces. To do this, we count the $T4$-*segments*, which are the free segments tangent to four balls with endpoints on two of those balls. Since there is a one-to-one correspondence between 0-faces and $T4$-segments when the objects are in adequate general position, this yields a bound on the expected number of vertices of the visibility complex. Note that since the balls are contained in $\mathcal{U}^+$, the $T4$-segments are also contained in $\mathcal{U}^+$.

Our main result is the following.

THEOREM 2.1. *The expected number of $T4$-segments amongst n uniformly distributed unit balls is $\Theta(n)$.*

We extend this result to the higher-dimensional faces of the complex.

THEOREM 2.2. *The expected size of the visibility complex of n uniformly distributed unit balls is $\Theta(n)$.*

We also present an $\Omega(n^2)$ worst-case lower bound on the number of $T4$-segments amongst n unit balls in $\mathbb{R}^3$ (see Proposition 6.1). In fact the lower bound holds for the number of k-faces of the visibility complex for all k between 0 and 4.

3. The expected number of $T4$-segments is at most linear. The general idea behind the proof of the upper bound of Theorem 2.1 is the following. For any ordered choice of four balls, we bound from above the probability that a line is tangent to these balls in the given order and is not occluded in between its contact points with the balls. Then we sum these probabilities over all ordered quadruples of balls and all potential tangent lines to these balls.

For any two points p and q, and positive real number α, let $\mathcal{H}(p,q,\alpha)$ denote the union of all the balls of radius α centered on the line segment pq (see Figure 3.1). We first show that a line is tangent to four balls B_i, B_j, B_k, and B_l in that order only if p_j and p_k are in $\mathcal{H}(p_i,p_l,2)$. Thus the volume of $\mathcal{H}(p_i,p_l,2) \cap \mathcal{U}$ gives an upper bound on the probability that a line tangent to the four balls in the given order exists.

[1] Formally, we consider the space of free segments quotiented by the equivalence relation that is the transitive and reflexive closure of the inclusion. In other words, two free segments are identified if they are both contained in the same maximal-free segment. This allows the cells of the partition to be connected.

We next show that a segment tangent to four balls B_i, B_j, B_k, and B_l in that order at points t_i, t_j, t_k, and t_l, respectively, is not occluded if and only if the centers of all remaining balls are outside or on the boundary of $\mathcal{H}(t_i, t_l, 1)$. The volume of $\mathcal{U} \setminus \mathcal{H}(t_i, t_l, 1)$ gives an upper bound on the probability that the tangent segment is not occluded. Thus, to get an upper bound on that probability, we need a lower bound on the volume of $\mathcal{H}(t_i, t_l, 1) \cap \mathcal{U}$.

To bound the probability that a $T4$-segment exists, we integrate over the distance between p_i and p_l, and over the distance from p_i to the boundary of the universe $\mathcal{U}$. This integral is split into three parts covering the cases where

(i) B_i and B_l are close to one another,

(ii) at least one of B_i and B_l is entirely inside the universe,

(iii) B_i and B_l are not close to one another and both are partially outside the universe.

In each case we overestimate the volume of $\mathcal{H}(p_i, p_l, 2) \cap \mathcal{U}$ and underestimate the volume of $\mathcal{H}(t_i, t_l, 1) \cap \mathcal{U}$. We apply the same general proof technique in each of the three cases. While case (ii) illustrates the main idea behind the proof (case (i) being a simplified version), extending this idea to case (iii) is technically challenging because of the difficulties caused by the boundary of the universe.

3.1. Definitions. Let $\mathcal{N}$ be the set of ordered 4-tuples (i, j, k, l) chosen from $\{1, 2, \ldots, n\}$ such that i, j, k, l are pairwise distinct. In our model, the probability that four centers are collinear is zero, so we may assume that any set of four balls admits at most 12 real common tangent lines [5, 13]. Moreover, the real common tangent lines correspond to the real solutions of a degree 12 system of equations. For any set of four balls we order arbitrarily the 12 solutions of the associated system.

Given four balls B_i, B_j, B_k, and B_l, we denote by $\mathcal{L}^{\omega}_{i,j,k,l}$, for ω in $\{1, \ldots, 12\}$, the event that the ωth solution of the system is real, that the corresponding real tangent line is tangent to the four balls B_i, B_j, B_k, and B_l in that order, and that p_i is not closer than p_l to the boundary of $\mathcal{U}$. Whenever $\mathcal{L}^{\omega}_{i,j,k,l}$ occurs, we denote the points of tangency of that line on B_i, B_j, B_k, B_l by t_i, t_j, t_k, t_l, respectively. Let $\delta^{\omega}_{i,j,k,l}$ be the event that $\mathcal{L}^{\omega}_{i,j,k,l}$ occurs and the line segment $t_i t_l$ is not occluded. Notice that if $\delta^{\omega}_{i,j,k,l}$ occurs, the balls B_i, B_j, B_k, B_l define a $T4$-segment, and that a $T4$-segment corresponds to a unique $\delta^{\omega}_{i,j,k,l}$.

Let $\mathbf{x}_{i,l}$ be the random variable representing the distance from p_i to p_l, and let $\mathbf{y}_i$ (resp., $\mathbf{y}_l$) be the random variable denoting the distance from p_i (resp., p_l) to the boundary of the universe.

In what follows, a random point p denotes a point chosen from the uniform distribution over $\mathcal{U}$.

3.2. The proof. There is a one-to-one correspondence between the $T4$-segments and the events $\delta^{\omega}_{i,j,k,l}$ that occur. We thus have the following straightforward lemma.

LEMMA 3.1. *The expected number of $T4$-segments amongst n uniformly distributed unit balls is*

$$\sum_{(i,j,k,l)\in\mathcal{N}} \sum_{\omega=1}^{12} \Pr(\delta^{\omega}_{i,j,k,l}).$$

We bound the probability $\Pr(\delta^{\omega}_{i,j,k,l})$ by integrating over the distance x between p_i and p_l and over the distance y from p_i to the boundary of the universe $\mathcal{U}$. The integral is split into three parts, covering the cases where (i) the balls B_i and B_l are close to one another, (ii) p_i is at distance at least 1 from the boundary of $\mathcal{U}$, and

(iii) the balls B_i and B_l are not close to one another and p_i is at distance less than 1 from the boundary of $\mathcal{U}$. Note that in the last case, if $\delta^{\omega}_{i,j,k,l}$ occurs, then both ball centers p_i and p_l are within distance 1 from the boundary of $\mathcal{U}$. Two balls are considered close to one another if their centers are closer than some sufficiently large constant; for technical reasons which are embedded in the proof of Proposition A.1, we actually define *close* to mean distance at most 6.

LEMMA 3.2. $\Pr(\delta^{\omega}_{i,j,k,l}) \leqslant I_{x\leqslant 6} + I_{y\geqslant 1} + I_{x>6,y<1}$, *where*

$$
\begin{aligned}
I_{x\leqslant 6} &= \int_{x=0}^{6} \Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x) \cdot \Pr(x \leqslant \mathbf{x}_{i,l} < x + dx),\\
I_{y\geqslant 1} &= \int_{x=0}^{2R} \Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i \geqslant 1) \cdot \Pr(x \leqslant \mathbf{x}_{i,l} < x + dx \mid \mathbf{y}_i \geqslant 1),\\
I_{x>6,y<1} &= \int_{x=6}^{2R} \int_{y=0}^{1} \Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_l \leqslant \mathbf{y}_i)\\
&\qquad \cdot \Pr((x \leqslant \mathbf{x}_{i,l} < x + dx) \cap (\mathbf{y}_l \leqslant \mathbf{y}_i) \mid \mathbf{y}_i = y)\\
&\qquad \cdot \Pr(y \leqslant \mathbf{y}_i < y + dy).
\end{aligned}
$$

Proof. By the total probability theorem (see [17]),

$$
\Pr(\delta^{\omega}_{i,j,k,l}) = \int_{x=0}^{2R} \Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x) \cdot \Pr(x \leqslant \mathbf{x}_{i,l} < x + dx).
$$

The integral can be split at $x = 6$, giving $I_{x\leqslant 6}$. Then applying the total probability theorem on what remains, we get

$$
\begin{aligned}
\int_{x=6}^{2R} \int_{y=0}^{R} &\Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y) \cdot \Pr((x \leqslant \mathbf{x}_{i,l} < x + dx) \mid \mathbf{y}_i = y)\\
&\cdot \Pr(y \leqslant \mathbf{y}_i < y + dy),
\end{aligned}
\tag{3.1}
$$

which can be split at $y = 1$. The part corresponding to y between 1 and R is equal to

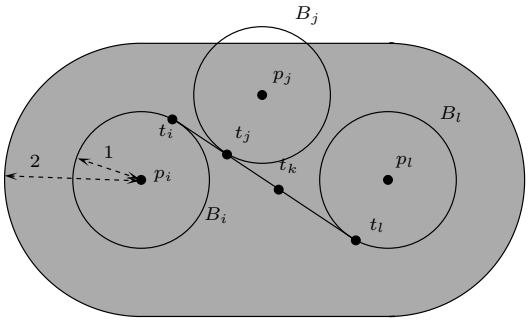
$$
\begin{aligned}
&\int_{x=6}^{2R} \int_{y=1}^{R} \Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_i \geqslant 1)\\
&\qquad \cdot \Pr((x \leqslant \mathbf{x}_{i,l} < x + dx) \mid \mathbf{y}_i = y,\ \mathbf{y}_i \geqslant 1) \cdot \Pr(y \leqslant \mathbf{y}_i < y + dy)\\
&\leqslant \int_{x=6}^{2R} \int_{y=0}^{R} \Pr(\delta^{\omega}_{i,j,k,l} \cap (x \leqslant \mathbf{x}_{i,l} < x + dx) \mid \mathbf{y}_i = y,\ \mathbf{y}_i \geqslant 1) \cdot \Pr(y \leqslant \mathbf{y}_i < y + dy).
\end{aligned}
$$

Applying the total probability theorem again, we get

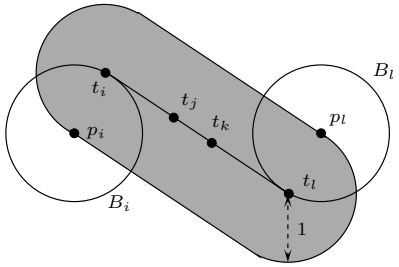
$$
\int_{x=6}^{2R} \Pr(\delta^{\omega}_{i,j,k,l} \cap (x \leqslant \mathbf{x}_{i,l} < x + dx) \mid \mathbf{y}_i \geqslant 1),
$$

which is less than $I_{y\geqslant 1}$. Consider now the part of (3.1) for y between 0 and 1. If $\mathbf{y}_l > \mathbf{y}_i$, then $\delta^{\omega}_{i,j,k,l}$ does not occur (by definition of $\mathcal{L}^{\omega}_{i,j,k,l}$); thus we have

$$
\begin{aligned}
&\Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y) \cdot \Pr((x \leqslant \mathbf{x}_{i,l} < x + dx) \mid \mathbf{y}_i = y)\\
&= \Pr(\delta^{\omega}_{i,j,k,l} \cap (x \leqslant \mathbf{x}_{i,l} < x + dx) \mid \mathbf{y}_i = y)\\
&= \Pr(\delta^{\omega}_{i,j,k,l} \cap (x \leqslant \mathbf{x}_{i,l} < x + dx) \cap (\mathbf{y}_l \leqslant \mathbf{y}_i) \mid \mathbf{y}_i = y)\\
&= \Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_l \leqslant \mathbf{y}_i)\\
&\quad \cdot \Pr((x \leqslant \mathbf{x}_{i,l} < x + dx) \cap (\mathbf{y}_l \leqslant \mathbf{y}_i) \mid \mathbf{y}_i = y).
\end{aligned}
$$



(a) $\mathcal{H}(p_i, p_l, 2)$



(b) $\mathcal{H}(t_i, t_l, 1)$

FIG. 3.1. *$\mathcal{H}(p_i, p_l, 2)$ and $\mathcal{H}(t_i, t_l, 1)$ are shown shaded.*

Thus, the part of (3.1) for y between 0 and 1 is equal to $I_{x>6,y<1}$. □

Let Ξ denote any of the following events: $(\mathbf{x}_{i,l} = x)$, $(\mathbf{x}_{i,l} = x, \mathbf{y}_i \geqslant 1)$, $(\mathbf{x}_{i,l} = x, \mathbf{y}_i = y, \mathbf{y}_l \leqslant \mathbf{y}_i)$. The next three lemmas are used to bound $\Pr(\delta^{\omega}_{i,j,k,l} \mid \Xi)$ appearing in the three integrals $I_{x\leqslant 6}$, $I_{y\geqslant 1}$, and $I_{x>6,y<1}$.

LEMMA 3.3. *If a line is tangent to four balls B_i, B_j, B_k, B_l in that order at t_i, t_j, t_k, t_l, respectively, then $p_j, p_k \in \mathcal{H}(p_i, p_l, 2)$. Also, the segment $t_i t_l$ is not occluded if and only if the interior of $\mathcal{H}(t_i, t_l, 1)$ does not contain the center of any other ball.*

Proof. Segment $t_i t_l$ is contained in $\mathcal{H}(p_i, p_l, 1)$. Since t_j and t_k belong to that segment, t_j and t_k are also in $\mathcal{H}(p_i, p_l, 1)$. Thus p_j, p_k are both in $\mathcal{H}(p_i, p_l, 2)$. See Figure 3.1(a).

The segment $t_i t_l$ is occluded if and only if some ball B_γ, $\gamma \neq i, j, k, l$, properly intersects it; that is, the center of B_γ lies in the interior of $\mathcal{H}(t_i, t_l, 1)$. See Figure 3.1(b). □

LEMMA 3.4. $\Pr(p \in \mathcal{H}(p_i, p_l, 2) \mid \Xi) \leqslant \frac{(3x+8)}{R^3}$.

Proof.

$$\Pr(p \in \mathcal{H}(p_i, p_l, 2) \mid \Xi) = \frac{\text{Volume of } \mathcal{H}(p_i, p_l, 2) \cap \mathcal{U}}{\text{Volume of } \mathcal{U}} \mid_{\Xi} \leqslant \frac{\text{Volume of } \mathcal{H}(p_i, p_l, 2)}{\text{Volume of } \mathcal{U}} \mid_{\Xi}.$$

When Ξ occurs, $\mathbf{x}_{i,l} = x$ and the volumes of $\mathcal{H}(p_i, p_l, 2)$ and $\mathcal{U}$ are $\frac{4\pi}{3}(3x + 8)$ and $\frac{4\pi}{3}R^3$, respectively. Thus

$$\Pr(p \in \mathcal{H}(p_i, p_l, 2) \mid \Xi) \leqslant \frac{3x + 8}{R^3}. \qquad \square$$

LEMMA 3.5. $\Pr(\delta^{\omega}_{i,j,k,l} \mid \Xi) \leqslant \frac{(3x+8)^2}{R^6} \cdot \Pr(p \notin \mathcal{H}(t_i, t_l, 1) \mid \mathcal{L}^{\omega}_{i,j,k,l},\ \Xi)^{n-4}$.

Proof. If $\delta^{\omega}_{i,j,k,l}$ occurs, then $\mathcal{L}^{\omega}_{i,j,k,l}$ necessarily occurs; thus

$$\Pr(\delta^{\omega}_{i,j,k,l} \mid \Xi) = \Pr(\delta^{\omega}_{i,j,k,l} \cap \mathcal{L}^{\omega}_{i,j,k,l} \mid \Xi) = \Pr(\mathcal{L}^{\omega}_{i,j,k,l} \mid \Xi) \cdot \Pr(\delta^{\omega}_{i,j,k,l} \mid \mathcal{L}^{\omega}_{i,j,k,l},\ \Xi).$$

By Lemma 3.3, $\Pr(\mathcal{L}^{\omega}_{i,j,k,l} \mid \Xi)$ is bounded by the probability that p_j and p_k belong to $\mathcal{H}(p_i, p_l, 2)$, given Ξ, and $\Pr(\delta^{\omega}_{i,j,k,l} \mid \mathcal{L}^{\omega}_{i,j,k,l})$ is equal to the probability that for all $\gamma \neq i, j, k, l$, point p_γ is outside $\mathcal{H}(t_i, t_l, 1)$, given Ξ. Since all the points are independently and identically drawn from the uniform distribution over $\mathcal{U}$, Lemma 3.4 yields the result. ∎

We consider the three integrals $I_{x\leqslant 6}$, $I_{y\geqslant 1}$, and $I_{x>6,y<1}$ in the following subsections and prove that each is bounded by $O\left(\frac{1}{n^3}\right)$. This will complete the proof of the upper bound of Theorem 2.1 since, by Lemmas 3.1 and 3.2, the expected number of $T4$-segments is less than $12\binom{n}{4}(I_{x\leqslant 6} + I_{y\geqslant 1} + I_{x>6,y<1})$.

3.2.1. B_i and B_l are close to one another. We prove here that $I_{x\leqslant 6}$ is $O\left(\frac{1}{n^3}\right)$. When B_i and B_l are close to one another, the probability that there exist two other balls, B_j and B_k, defining a line tangent to B_i, B_j, B_k, B_l in that order is small enough that we do not need to consider occlusions in order to get the bound we want.

We first bound the term $\Pr(x \leqslant \mathbf{x}_{i,l} < x + dx)$ appearing in the integral $I_{x\leqslant 6}$.

LEMMA 3.6. $\Pr(x \leqslant \mathbf{x}_{i,l} < x + dx) \leqslant \frac{3x^2}{R^3} dx$.

Proof. When p_i is given, p_l must belong to a spherical shell between two spheres of center p_i and radii x and $x + dx$. The probability $\Pr(x \leqslant \mathbf{x}_{i,l} < x + dx)$, if p_i is known, is exactly the volume of the part of the spherical shell inside $\mathcal{U}$ divided by the volume of $\mathcal{U}$. The volume of the part of the spherical shell inside $\mathcal{U}$ is bounded from above by the volume of the spherical shell, which is $4\pi x^2 dx$. Since the volume of $\mathcal{U}$ is $\frac{4}{3}\pi R^3$ we get the claimed bound. (The exact value of $\Pr(x \leqslant \mathbf{x}_{i,l} < x + dx)$ is actually given in [15, 22], but the above approximate bound is enough for our purposes.) ∎

PROPOSITION 3.7. *$I_{x\leqslant 6}$ is $O\left(\frac{1}{n^3}\right)$.*

Proof. Recall that (see Lemma 3.2)

$$I_{x\leqslant 6} = \int_{x=0}^{6} \Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x) \cdot \Pr(x \leqslant \mathbf{x}_{i,l} < x + dx).$$

By Lemma 3.5,

$$\begin{aligned}\Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x) &\leqslant \frac{(3x+8)^2}{R^6} \cdot \Pr(p \notin \mathcal{H}(t_i, t_l, 1) \mid \mathbf{x}_{i,l} = x,\ \mathcal{L}^{\omega}_{i,j,k,l})^{n-4} \\ &\leqslant \frac{(3x+8)^2}{R^6}.\end{aligned}$$

It thus follows from Lemma 3.6 that

$$I_{x\leqslant 6} \leqslant \int_{x=0}^{6} \frac{(3x+8)^2}{R^6} \cdot \frac{3x^2}{R^3} dx = \frac{\mu^3}{n^3} \int_{x=0}^{6} 3x^2(3x+8)^2 dx = O\left(\frac{1}{n^3}\right). \quad ∎$$

3.2.2. B_i is entirely inside $\mathcal{U}$. For the integral $I_{y\geqslant 1}$, occlusions must be taken into account. To this aim, we bound from below the volume of $\mathcal{H}(t_i, t_l, 1) \cap \mathcal{U}$ in the following lemma.

LEMMA 3.8. *When $\mathcal{L}^{\omega}_{i,j,k,l}$ occurs and $\mathbf{y}_i \geqslant 1$, the volume of $\mathcal{H}(t_i, t_l, 1) \cap \mathcal{U}$ is greater than $\frac{\pi}{12}\mathbf{x}_{i,l}$.*

Proof. Let K be the ball having diameter p_it_i. Note that K and p_l are both contained in $\mathcal{U}$ and in $\mathcal{H}(t_i,t_l,1)$. The convex hull of p_l and K is thus contained in $\mathcal{H}(t_i,t_l,1)\cap\mathcal{U}$, and its volume is larger than half the volume of the ball K, $\frac{\pi}{12}$, plus the volume of a cone of apex p_l, of base a disk whose boundary is a great circle of K, and of height greater than $\mathbf{x}_{i,l}-1$. The volume of that cone is at least $\frac{1}{3}\frac{\pi}{2^2}(\mathbf{x}_{i,l}-1)=\frac{\pi}{12}\mathbf{x}_{i,l}-\frac{\pi}{12}$. □

We now bound the probability that a tangent line segment t_it_l is not occluded by any of the other $n-4$ balls, given that the line segment t_it_l exists and the ball B_i is entirely contained in $\mathcal{U}$.

LEMMA 3.9. $\Pr\left(p\notin\mathcal{H}(t_i,t_l,1)\mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i\geqslant 1,\ \mathcal{L}^{\omega}_{i,j,k,l}\right)^{n-4}<55\exp\left(-\frac{\mu\,x}{16}\right)$.

Proof. First notice that

$$\Pr\left(p\notin\mathcal{H}(t_i,t_l,1)\mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i\geqslant 1,\ \mathcal{L}^{\omega}_{i,j,k,l}\right) = 1-\frac{\text{Volume of }\mathcal{H}(t_i,t_l,1)\cap\mathcal{U}}{\text{Volume of }\mathcal{U}}\Big|_{\mathbf{x}_{i,l}=x,\ \mathbf{y}_i\geqslant 1,\ \mathcal{L}^{\omega}_{i,j,k,l}}.$$

By Lemma 3.8, the volume of $\mathcal{H}(t_i,t_l,1)\cap\mathcal{U}$ is bounded from below by $\frac{\pi}{12}x$. Since the volume of $\mathcal{U}$ is $\frac{4}{3}\pi R^3$, we get

$$\Pr\left(p\notin\mathcal{H}(t_i,t_l,1)\mid \mathbf{x}_{i,l}=x,\mathbf{y}_i\geqslant 1,\ \mathcal{L}^{\omega}_{i,j,k,l}\right)^{n-4}<\left(1-\frac{x}{16R^3}\right)^{n-4}.$$

For any $0\leqslant t\leqslant 1$, we have $(1-t)\leqslant e^{-t}$; thus

$$(1-t)^{n-4}\leqslant e^{-t(n-4)}=e^{-tn}e^{4t}\leqslant e^4e^{-tn}<55\,e^{-tn}.$$

Now $0\leqslant x\leqslant 2R$ and $R\geqslant 1$ since B_i is entirely inside $\mathcal{U}$. Thus $0\leqslant\frac{x}{16R^3}\leqslant\frac{1}{8R^2}\leqslant 1$ and

$$\Pr\left(p\notin\mathcal{H}(t_i,t_l,1)\mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i\geqslant 1,\ \mathcal{L}^{\omega}_{i,j,k,l}\right)^{n-4}<55\exp\left(-\frac{n\,x}{16R^3}\right) = 55\exp\left(-\frac{\mu\,x}{16}\right).$$ □

The following proposition now bounds the integral $I_{y\geqslant 1}$.

PROPOSITION 3.10. *$I_{y\geqslant 1}$ is $O\left(\frac{1}{n^3}\right)$.*

Proof. Recall that

$$I_{y\geqslant 1}=\int_{x=0}^{2R}\Pr(\delta^{\omega}_{i,j,k,l}\mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i\geqslant 1)\cdot\Pr(x\leqslant\mathbf{x}_{i,l}<x+dx\mid \mathbf{y}_i\geqslant 1).$$

By Lemmas 3.5 and 3.9 we have

$$\Pr(\delta^{\omega}_{i,j,k,l}\mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i\geqslant 1)\leqslant\frac{(3x+8)^2}{R^6}\cdot 55\exp\left(-\frac{\mu\,x}{16}\right).$$

Similarly as in Lemma 3.6 we have

$$\Pr(x\leqslant\mathbf{x}_{i,l}<x+dx\mid \mathbf{y}_i\geqslant 1)\leqslant\frac{3x^2}{R^3}dx.$$

Thus we get

$$\begin{aligned} I_{y\geqslant 1} &\leqslant\int_{x=0}^{2R}\frac{(3x+8)^2}{R^6}\cdot 55\exp\left(-\frac{\mu\,x}{16}\right)\cdot\frac{3x^2}{R^3}dx\\ &\leqslant\frac{\mu^3}{n^3}\int_{x=0}^{+\infty}3x^2(3x+8)^2\cdot 55\exp\left(-\frac{\mu\,x}{16}\right)dx. \end{aligned}$$

Changing $\frac{\mu x}{16}$ by z we get integrals of the kind

$$\int_0^\infty z^r \exp(-z)dz,$$

which is bounded by a constant, and thus $I_{y\geqslant 1}$ is $O\left(\frac{1}{n^3}\right)$. □

3.2.3. B_i and B_l are not close to one another and B_i is partially outside $\mathcal{U}$. The only remaining task is to bound the integral $I_{x>6,y<1}$. As in the previous case, we need to bound from below the volume of $\mathcal{H}(t_i,t_l,1)\cap\mathcal{U}$. Here, however, the tangent t_it_l can be entirely outside $\mathcal{U}$, so the bound of Lemma 3.8 does not apply, and a more intricate proof is needed. We need to distinguish two cases depending on the distance of segment t_it_l from O, the center of $\mathcal{U}$.

To this aim, we introduce two new types of events. For any $s\in\mathbb{R}$, let $\mathcal{F}^\omega_{i,j,k,l}(s)$ (resp., $\mathcal{N}^\omega_{i,j,k,l}(s)$) be the event that $\mathcal{L}^\omega_{i,j,k,l}$ occurs and the line segment t_it_l is at distance greater (resp., less) than $R+1-s$ from O. For reasons that will become clear in the proof of Lemma 3.13, we consider $s=y^{\frac{2}{3}}$.

The next five lemmas are used to bound the first term of the integral $I_{x>6,y<1}$.

LEMMA 3.11. *For any random point p in $\mathcal{U}$,* $\Pr(\delta^\omega_{i,j,k,l} \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i)$ *is equal to*

$$\begin{aligned}
&\Pr\left(\mathcal{F}^\omega_{i,j,k,l}(y^{\frac{2}{3}}) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i\right)\\
&\cdot\Pr\left(p\notin\mathcal{H}(t_i,t_l,1) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i,\ \mathcal{F}^\omega_{i,j,k,l}(y^{\frac{2}{3}})\right)^{n-4}\\
&+\Pr\left(\mathcal{N}^\omega_{i,j,k,l}(y^{\frac{2}{3}}) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i\right)\\
&\cdot\Pr\left(p\notin\mathcal{H}(t_i,t_l,1) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i,\ \mathcal{N}^\omega_{i,j,k,l}(y^{\frac{2}{3}})\right)^{n-4}.
\end{aligned}$$

Proof. $\delta^\omega_{i,j,k,l}$ implies $\mathcal{L}^\omega_{i,j,k,l}$, which can be split into $\mathcal{F}^\omega_{i,j,k,l}(y^{\frac{2}{3}})$, $\mathcal{N}^\omega_{i,j,k,l}(y^{\frac{2}{3}})$, and the event that $\mathcal{L}^\omega_{i,j,k,l}$ occurs and the line segment t_it_l is at distance exactly $R+1-y^{\frac{2}{3}}$ from O. This later event occurs with probability 0; thus

$$\begin{aligned}
\Pr(\delta^\omega_{i,j,k,l} \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i)&\\
&=\Pr(\delta^\omega_{i,j,k,l}\cap\mathcal{F}^\omega_{i,j,k,l}(y^{\frac{2}{3}}) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i)\\
&\quad+\Pr(\delta^\omega_{i,j,k,l}\cap\mathcal{N}^\omega_{i,j,k,l}(y^{\frac{2}{3}}) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i),
\end{aligned}$$

which can be expanded into

$$\begin{aligned}
&\Pr(\mathcal{F}^\omega_{i,j,k,l}(y^{\frac{2}{3}}) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i)\\
&\cdot\Pr(\delta^\omega_{i,j,k,l} \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i,\ \mathcal{F}^\omega_{i,j,k,l}(y^{\frac{2}{3}}))\\
&+\Pr(\mathcal{N}^\omega_{i,j,k,l}(y^{\frac{2}{3}}) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i)\\
&\cdot\Pr(\delta^\omega_{i,j,k,l} \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i,\ \mathcal{N}^\omega_{i,j,k,l}(y^{\frac{2}{3}})).
\end{aligned}$$

When $\mathcal{F}^\omega_{i,j,k,l}(y^{\frac{2}{3}})$ occurs, the probability

$$\Pr(\delta^\omega_{i,j,k,l} \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i,\ \mathcal{F}^\omega_{i,j,k,l}(y^{\frac{2}{3}}))$$

is the probability that the tangent is not occluded; that is, p_γ does not belong to $\mathcal{H}(t_i,t_l,1)$ for all the $n-4$ values of $\gamma\neq i,j,k,l$. The same argument holds for $\mathcal{N}^\omega_{i,j,k,l}(y^{\frac{2}{3}})$. Since the p_γ are independent, we get the result. □

In order to bound the two terms in Lemma 3.11,

$$\Pr\left(p \notin \mathcal{H}(t_i,t_l,1) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l \leqslant \mathbf{y}_i,\ \mathcal{F}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})\right)^{n-4}$$

and

$$\Pr\left(p \notin \mathcal{H}(t_i,t_l,1) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l \leqslant \mathbf{y}_i,\ \mathcal{N}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})\right)^{n-4},$$

we need to bound the volume of $\mathcal{H}(t_i,t_l,1)\cap\mathcal{U}$ from below.

LEMMA 3.12. *When $\mathbf{x}_{i,l} \geqslant 6$, $\mathbf{y}_l \leqslant \mathbf{y}_i \leqslant 1$, $\mathcal{L}^{\omega}_{i,j,k,l}$ occurs and segment t_it_l is at distance less than $R+1-s$, $0 \leqslant s \leqslant 1$, from the center of $\mathcal{U}$, then the volume of $\mathcal{H}(t_i,t_l,1)\cap\mathcal{U}$ is larger than $\frac{1}{6\sqrt{2}}\,(\mathbf{x}_{i,l}-5)\,s\,\sqrt{s}$.*

Proof. We give here the idea of the proof; full details can be found in Appendix A. Let t be the closest point on segment t_it_l from O, and let D be a unit radius disk centered at t in a plane containing O, the center of $\mathcal{U}$. We define a quadrilateral with vertices a,b,a',b' such that a and a' are the closest and the farthest points, respectively, in $D\cap\mathcal{U}$ from O, and b and b' are the points of intersection of ∂D and the perpendicular bisector of segment aa' (see Figure 3.2). Let v be equal to $R+1$ minus the distance from O to segment t_it_l. We prove that the convex hull of a,b,a',b', and p_l, which is included in $\mathcal{H}(t_i,t_l,1)\cap\mathcal{U}$, has volume greater than $\frac{1}{6\sqrt{2}}\,(\mathbf{x}_{i,l}-5)\,\min(2\sqrt{2},v\sqrt{v})$. It follows that, for any $0\leqslant s\leqslant 1$, if segment t_it_l is at distance less than $R+1-s$ from O, then $v\geqslant s$, and the volume of $\mathcal{H}(t_i,t_l,1)\cap\mathcal{U}$ is greater than $\frac{1}{6\sqrt{2}}\,(\mathbf{x}_{i,l}-5)\,s\,\sqrt{s}$. ∎

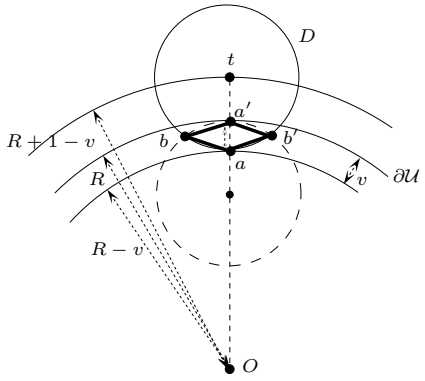


FIG. 3.2. *For the sketch of the proof of Lemma 3.12 ($v\in(0,1)$).*

LEMMA 3.13. *For any random point p in $\mathcal{U}$, $x\geqslant 6$ and $0\leqslant y\leqslant 1$,*

$$\Pr\left(p \notin \mathcal{H}(t_i,t_l,1) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l \leqslant \mathbf{y}_i,\ \mathcal{F}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})\right)^{n-4} < 55\exp\left(-\frac{\mu\,(x-5)\,y^2}{8\sqrt{2}\,\pi}\right)$$

and

$$\Pr\left(p \notin \mathcal{H}(t_i,t_l,1) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l \leqslant \mathbf{y}_i,\ \mathcal{N}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})\right)^{n-4} < 55\exp\left(-\frac{\mu\,(x-5)\,y}{8\sqrt{2}\,\pi}\right).$$

Furthermore, if $x \geqslant 6\sqrt{R}$, *then*

$$\Pr\left(p \notin \mathcal{H}(t_i,t_l,1) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l \leqslant \mathbf{y}_i,\ \mathcal{N}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})\right)^{n-4} < 55\exp\left(-\frac{\mu\,(x-5)}{8\,\sqrt{2}\,\pi}\right).$$

Proof. Let $\mathbf{x}_{i,l}=x$, $\mathbf{y}_i=y$, and suppose first that event $\mathcal{F}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})$ occurs. Since p_i is at distance $R-y$ from O, the segment t_it_l is at distance less than $R+1-y$ from O, and thus, by Lemma 3.12, the volume of $\mathcal{H}\,(t_i,t_l,1)\cap\mathcal{U}$ is greater than $\frac{1}{6\sqrt{2}}\,(x-5)\,y\,\sqrt{y}$, which is bigger than $\frac{1}{6\sqrt{2}}\,(x-5)\,y^2$ since $0\leqslant y\leqslant 1$ (we bound $y\sqrt{y}$ from below by y^2 only so that we can actually compute the integral $\mathcal{I}_1$ in the proof of Proposition 3.18). We now follow the proof of Lemma 3.9, except that the volume of $\mathcal{H}\,(t_i,t_l,1)\cap\mathcal{U}$ is now bounded from below by $\frac{1}{6\sqrt{2}}\,(x-5)\,y^2$ instead of $\frac{\pi}{12}\,x$. We get

$$\Pr\left(p \notin \mathcal{H}(t_i,t_l,1) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l \leqslant \mathbf{y}_i,\ \mathcal{F}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})\right)^{n-4} < 55\exp\left(-\frac{\mu\,(x-5)\,y^2}{8\,\sqrt{2}\,\pi}\right).$$

When $\mathcal{N}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})$ occurs, the segment t_it_l is at distance less than $R+1-y^{\frac{2}{3}}$ from O, and thus, by Lemma 3.12, the volume of $\mathcal{H}\,(t_i,t_l,1)\cap\mathcal{U}$ is bounded from below by $\frac{1}{6\sqrt{2}}\,(x-5)\,y^{\frac{2}{3}}\,\sqrt{y^{\frac{2}{3}}}=\frac{1}{6\sqrt{2}}\,(x-5)\,y$. Then, as before, we get

$$\Pr\left(p \notin \mathcal{H}(t_i,t_l,1) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l \leqslant \mathbf{y}_i,\ \mathcal{N}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})\right)^{n-4} < 55\exp\left(-\frac{\mu\,(x-5)\,y}{8\,\sqrt{2}\,\pi}\right).$$

Now, if $x \geqslant 6\sqrt{R}$, the length of the tangent t_it_l is at least $6\sqrt{R}-2$. Since $x\geqslant 6$, $R>3$ and a simple computation show that $6\sqrt{R}-2$ is bigger than $2\sqrt{2R+1}$, which is the length of the longest line segment that may entirely lie inside $\mathcal{U}^+\setminus\mathcal{U}$. Thus $dist(O,t_it_l)\leqslant R = R+1-s$ with $s=1$, and, by Lemma 3.12, the volume of $\mathcal{H}(t_i,t_l,1)\cap\mathcal{U}$ is greater than $\frac{1}{6\sqrt{2}}\,(x-5)$. Then, as before, we get

$$\Pr\left(p \notin \mathcal{H}(t_i,t_l,1) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l \leqslant \mathbf{y}_i,\ \mathcal{N}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})\right)^{n-4} < 55\exp\left(-\frac{\mu\,(x-5)}{8\,\sqrt{2}\,\pi}\right). \qquad \square$$

LEMMA 3.14. $\Pr(\mathcal{N}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}}) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i) \leqslant \frac{(3x+8)^2}{R^6}$.

Proof. The event $\mathcal{N}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})$ occurs only if $\mathcal{L}^{\omega}_{i,j,k,l}$ occurs. The result thus follows since, by Lemmas 3.3 and 3.4, $\Pr(\mathcal{L}^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y,\ \mathbf{y}_l\leqslant\mathbf{y}_i) \leqslant \frac{(3x+8)^2}{R^6}$. $\square$

LEMMA 3.15. *If* $y<1$, *then*

$$\Pr\left(\mathcal{F}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}}) \mid \mathbf{x}_{i,l}=x,\ \mathbf{y}_i=y\right) \leqslant 81\,\pi^2\,\frac{(x+6)^2y^2}{R^6}.$$

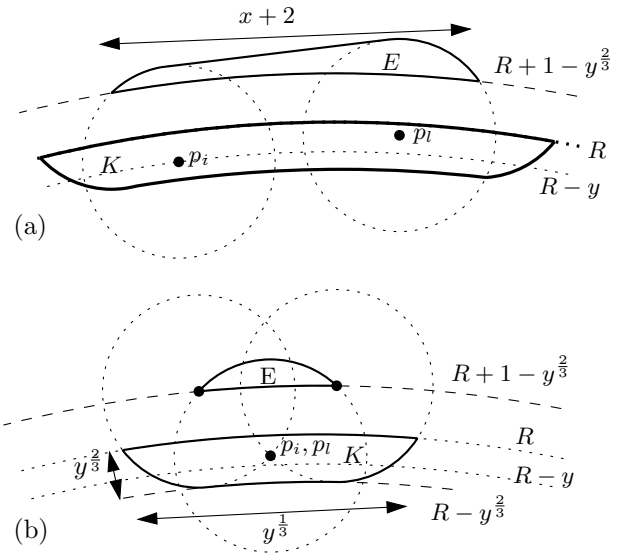


FIG. 3.3. *For the sketch of the proof of Lemma* 3.15.

Proof. A "far" tangent $t_i t_l$ is at distance at least $R+1-y^{\frac{2}{3}}$ from the center O of $\mathcal{U}$. Such a segment also lies in $\mathcal{H}(p_i, p_l, 1)$. Let E be the part of $\mathcal{H}(p_i, p_l, 1)$ lying outside of the sphere of radius $R+1-y^{\frac{2}{3}}$ and center O. See Figure 3.3(a). Now, both p_j and p_k must be in the region inside $\mathcal{U}$ and within distance 1 from E. Denote this region by K. Then

$$\Pr\left(\mathcal{F}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}}) \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_l \leqslant \mathbf{y}_i\right) \leqslant \left(\frac{\text{Volume of } K}{\text{Volume of } \mathcal{U}}\right)^2.$$

By Proposition B.1, which we prove in Appendix B, the volume of K is bounded from above by $12\,\pi^2\,(x+6)\,y$, which yields the result. Here we give the intuition of the proof. Refer to Figure 3.3. First notice that the "length" of K is at most $x+4$. Since K is enclosed in between a sphere of radius R and one of radius $R-y^{\frac{2}{3}}$, its "height" is at most $y^{\frac{2}{3}}$. For the "width," consider Figure 3.3(b), which shows a cross-section of K taken with a plane through O and perpendicular to $p_i p_l$. The "width" of K is no more than two times the "width" of E. The "height" of E can be bounded by some constant times $y^{\frac{2}{3}}$; thus its "width" can be bounded by some constant times $\sqrt{y^{\frac{2}{3}}} = y^{\frac{1}{3}}$. Thus, intuitively, the volume of K is smaller than $(x+4)y^{\frac{2}{3}}y^{\frac{1}{3}} = (x+4)y$, up to a constant, and the result follows. ∎

We now bound the two last terms of the integral $I_{x>6,y<1}$.

LEMMA 3.16. $\Pr(y \leqslant \mathbf{y}_i < y+dy) \leqslant \frac{3\,dy}{R}$.

Proof. The event $(y \leqslant \mathbf{y}_i < y+dy)$ occurs only if p_i lies in the spherical shell delimited by the two spheres centered at O of radii $R-y$ and $R-y-dy$ whose volume is smaller than $4\,\pi\,R^2\,dy$. Dividing by the volume of $\mathcal{U}$ proves the result. ∎

LEMMA 3.17. *For* $6 \leqslant x \leqslant 2R$ *and* $y \leqslant 1$, *we have*

$$\Pr((x \leqslant \mathbf{x}_{i,l} < x+dx) \cap (\mathbf{y}_l \leqslant \mathbf{y}_i) \mid \mathbf{y}_i = y) \leqslant \frac{6\,x\,y\,dx}{R^3}.$$

Proof. The probability $\Pr((x \leqslant \mathbf{x}_{i,l} < x+dx) \cap (\mathbf{y}_l \leqslant \mathbf{y}_i) \mid \mathbf{y}_i = y)$ is equal to the volume of the region (shown in grey in Figure 3.4), which is the intersection of the

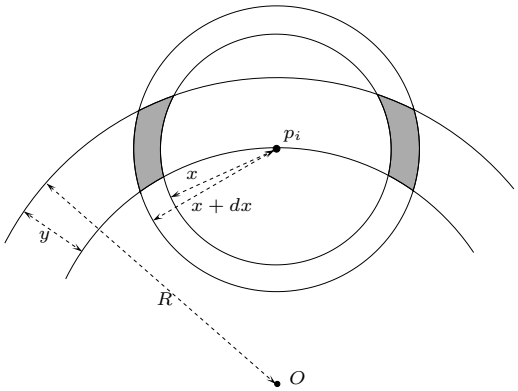


FIG. 3.4. *For the proof of Lemma* 3.17.

region in between the two spheres centered at p_i and of radii x and $x + dx$, and the region in between the two spheres centered at O and of radii R and $R - y$, divided by the volume of $\mathcal{U}$. We prove in Proposition C.1 in Appendix C that the volume of that region is at most $8\,\pi\,x\,y\,dx$. Roughly speaking, the volume bounded by the four spheres is at most $8\,\pi\,x\,y\,dx$ because its "thickness" is dx, its "height" is y, and its "radius" is x. Dividing by the volume of $\mathcal{U}$ proves the result. □

We can now bound the integral $I_{x>6,y<1}$ of Lemma 3.2.

PROPOSITION 3.18. *$I_{x>6,y<1}$ is $O\left(\frac{1}{n^3}\right)$.*

Proof. Recall that

$$\begin{aligned} I_{x>6,y<1} = \int_{x=6}^{2R}\int_{y=0}^{1} & \Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_l \leqslant \mathbf{y}_i) \\ & \cdot \Pr((x \leqslant \mathbf{x}_{i,l} < x + dx) \cap (\mathbf{y}_l \leqslant \mathbf{y}_i) \mid \mathbf{y}_i = y) \\ & \cdot \Pr(y \leqslant \mathbf{y}_i < y + dy). \end{aligned}$$

By Lemmas 3.16 and 3.17, we get

$$I_{x>6,y<1} \leqslant \int_{x=6}^{2R}\int_{y=0}^{1} \Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_l \leqslant \mathbf{y}_i) \cdot \frac{6\,x\,y\,dx}{R^3} \cdot \frac{3\,dy}{R}.$$

By Lemma 3.11, $\Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_l \leqslant \mathbf{y}_i)$ is equal to

$$\begin{aligned} & \Pr\left(\mathcal{F}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}}) \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_l \leqslant \mathbf{y}_i\right) \\ & \cdot \Pr\left(p \notin \mathcal{H}(t_i, t_l, 1) \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_l \leqslant \mathbf{y}_i,\ \mathcal{F}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})\right)^{n-4} \\ & + \Pr\left(\mathcal{N}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}}) \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_l \leqslant \mathbf{y}_i\right) \\ & \cdot \Pr\left(p \notin \mathcal{H}(t_i, t_l, 1) \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_l \leqslant \mathbf{y}_i,\ \mathcal{N}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}})\right)^{n-4}. \end{aligned}$$

We split the integral at $x = 6\sqrt{R}$. When $x \geqslant 6\sqrt{R}$, the distance from O to the tangent $t_i t_l$ is less than R (see the proof of Lemma 3.13), which is less than $R + 1 - y^{\frac{2}{3}}$ for any y in $(0, 1)$. Thus, for any $x \geqslant 6\sqrt{R}$ and $y \in (0, 1)$, the probability $\Pr\left(\mathcal{F}^{\omega}_{i,j,k,l}(y^{\frac{2}{3}}) \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_l \leqslant \mathbf{y}_i\right)$ is equal to 0. It then follows from Lemmas

3.13, 3.14, and 3.15 that $I_{x>6,y<1} \leqslant \mathcal{I}_1 + \mathcal{I}_2 + \mathcal{I}_3$ with

$$\mathcal{I}_1 = \int_{x=6}^{6\sqrt{R}} \int_{y=0}^{1} 81\,\pi^2 \frac{(x+6)^2 y^2}{R^6} \cdot 55 \exp\left(-\frac{\mu\,(x-5)\,y^2}{8\sqrt{2}\,\pi}\right) \cdot \frac{6\,x\,y\,dx}{R^3} \cdot \frac{3\,dy}{R},$$

$$\mathcal{I}_2 = \int_{x=6}^{6\sqrt{R}} \int_{y=0}^{1} \frac{(3x+8)^2}{R^6} \cdot 55 \exp\left(-\frac{\mu\,(x-5)\,y}{8\sqrt{2}\,\pi}\right) \cdot \frac{6\,x\,y\,dx}{R^3} \cdot \frac{3\,dy}{R},$$

$$\mathcal{I}_3 = \int_{x=6\sqrt{R}}^{2R} \int_{y=0}^{1} \frac{(3x+8)^2}{R^6} \cdot 55 \exp\left(-\frac{\mu\,(x-5)}{8\sqrt{2}\,\pi}\right) \cdot \frac{6\,x\,y\,dx}{R^3} \cdot \frac{3\,dy}{R}.$$

Changing $\frac{\mu\,(x-5)}{8\sqrt{2}\,\pi}$ by z in the three integrals and y^2 by y' in $\mathcal{I}_1$, we get

$$\mathcal{I}_1 \leqslant \frac{K}{R^{10}} \sum_{u=0}^{u=3} \int_{z=0}^{c\sqrt{R}} \int_{y'=0}^{1} z^u\, y'\, \exp(-z\,y')\,dz\,dy',$$

$$\mathcal{I}_2 \leqslant \frac{K}{R^{10}} \sum_{u=0}^{u=3} \int_{z=0}^{c\sqrt{R}} \int_{y=0}^{1} z^u\, y\, \exp(-z\,y)\,dz\,dy,$$

$$\mathcal{I}_3 \leqslant \frac{K}{R^{10}} \sum_{u=0}^{u=3} \int_{z=0}^{\infty} \int_{y=0}^{1} z^u\, y\, \exp(-z)\,dz\,dy,$$

where K and c are some positive constants.

Note first that $\mathcal{I}_3$ is bounded from above by $\frac{K}{R^{10}} \sum_{u=0}^{u=3} \int_{z=0}^{\infty} z^u \exp(-z)\,dz$. These integrals are bounded by a constant; thus $\mathcal{I}_3$ is $O\left(\frac{1}{R^{10}}\right)$.

To bound the integrals $\mathcal{I}_1$ and $\mathcal{I}_2$, we now compute the integral

$$\int_{z=0}^{A} \int_{y=0}^{1} z^u\, y\, \exp(-z\,y)\,dz\,dy \tag{3.2}$$

for $u \in \{0, \ldots, 3\}$ and $A > 0$, for example with Maple [14]. For $u = 0$ it is equal to

$$\frac{\exp(-A) + A - 1}{A}. \tag{3.3}$$

For $u = 1$, the integral (3.2) is equal to

$$\exp(-A) + \ln A + Ei(1, A) + \gamma - 1, \tag{3.4}$$

where $Ei(1, A)$ denotes the exponential integral $\int_{t=1}^{\infty} \frac{\exp(-A\,t)}{t}\,dt$ and γ denotes Euler's constant. Finally, for $u = 2$ or 3, the integral (3.2) is equal to

$$\exp(-A)\,P_1(A, u-1) + P_2(A, u-1), \tag{3.5}$$

where $P_i(A, u-1)$ denotes a polynomial of degree $u-1$ in A.

When A tends to ∞, (3.3) tends to 1, (3.4) is equivalent to $\ln A$ (since $Ei(1, A)$ tends to 0), and (3.5) is equivalent to the leading monomial of $P_2(A, u-1)$, which is of degree $u - 1 \leqslant 2$. This guarantees that for $A = c\sqrt{R}$ and $u \in \{0, \ldots, 3\}$, the integral (3.2) is $O(R)$. It follows that $\mathcal{I}_1$ and $\mathcal{I}_2$ are $O\left(\frac{1}{R^9}\right)$.

Since $R^3 = n/\mu$, we get that $I_{x>6,y<1} \leqslant \mathcal{I}_1 + \mathcal{I}_2 + \mathcal{I}_3 = O\left(\frac{1}{R^9}\right) = O\left(\frac{1}{n^3}\right)$. □

We can now conclude the proof that the expected number of $T4$-segments is $O(n)$, because, by Lemmas 3.1 and 3.2, and Propositions 3.7, 3.10, and 3.18, the expected number of $T4$-segments is smaller than

$$\sum_{(i,j,k,l)\in\mathcal{N}} \sum_{\omega=1}^{12} \left(O\left(\frac{1}{n^3}\right) + O\left(\frac{1}{n^3}\right) + O\left(\frac{1}{n^3}\right)\right) = O\left(n\right).$$

4. The expected number of $T4$-segments is at least linear. In this section, we prove that the expected number of $T4$-segments amongst n uniformly distributed unit balls is $\Omega(n)$. To do this, we bound from below the probability that four given balls have a given $T4$-segment. The key step is to give a condition on the relative positions of four unit balls that guarantees that they have exactly 12 common tangent lines. We use here the notation as defined in section 3.1.

LEMMA 4.1. *Let e be a real number satisfying $\frac{4\sqrt{2}}{3} < e < 2$, and let the radius R of $\mathcal{U}$ be strictly greater than e. There exists an $\epsilon > 0$ such that for any point $p \in \mathcal{U}$, there exist three balls $\Gamma_1(p)$, $\Gamma_2(p)$, $\Gamma_3(p)$ of radius ϵ contained in $\mathcal{U}$ and satisfying the following conditions:*

- *p and the centers of the $\Gamma_i(p)$ form a regular tetrahedron with edges of length e, and*
- *for any triple of points (p_1, p_2, p_3), p_i taken from $\Gamma_i(p)$, the four unit balls centered at p, p_1, p_2, and p_3 have exactly 12 distinct tangent lines.*

Proof. Macdonald, Pach, and Theobald proved [13, Lemma 3] that four unit balls centered on the vertices of a regular tetrahedron with edges of length e, $\frac{4\sqrt{2}}{3} < e < 2$, have exactly 12 distinct real common tangent lines. Moreover, these 12 tangent lines correspond to the 12 real roots of a system of equations of degree 12; thus each tangent line corresponds to a *simple* root of that system of equations. It thus follows that for any sufficiently small perturbation of the four ball centers, the four perturbed balls still have 12 real common tangent lines. Let $\epsilon > 0$ be such that the four ball centers can move distance ϵ in any direction while keeping 12 distinct common tangents.

Now, for any point $p \in \mathcal{U}$, consider a regular tetrahedron with edge length e having p as a vertex and such that the other vertices are at distance at least ϵ from the boundary of $\mathcal{U}$; for example, we can choose the other three vertices on a plane perpendicular to the segment Op. Let $\Gamma_1(p)$, $\Gamma_2(p)$, and $\Gamma_3(p)$ be the balls of radius ϵ centered at the vertices, distinct from p, of that tetrahedron. By the previous reasoning, for any $q \in \Gamma_1(p)$, $r \in \Gamma_2(p)$, and $s \in \Gamma_3(p)$, the four unit balls centered at p, q, r, and s have exactly 12 tangents. ∎

Now, by Lemma 3.1, the expected number of $T4$-segments is

$$\sum_{(i,j,k,l)\in\mathcal{N}} \sum_{\omega=1}^{12} \Pr(\delta^{\omega}_{i,j,k,l}).$$

Thus we need only to bound from below the probability that the event $\delta^{\omega}_{i,j,k,l}$ occurs.

LEMMA 4.2. $\Pr(\delta^{\omega}_{i,j,k,l})$ *is* $\Omega\left(\frac{1}{n^3}\right)$.

Proof. Assume that $n > 8\mu$ so that the radius $R = \sqrt[3]{n/\mu}$ of $\mathcal{U}$ is larger than 2, and let $T(p)$ be the set $\Gamma_1(p) \times \Gamma_2(p) \times \Gamma_3(p)$, where $\Gamma_i(p)$ and e are defined as in Lemma 4.1. First, note that

$$\begin{aligned}\Pr(\delta^{\omega}_{i,j,k,l}) &\geqslant \Pr(\delta^{\omega}_{i,j,k,l} \cap (p_i, p_j, p_k) \in T(p_l)) \\ &= \Pr((p_i, p_j, p_k) \in T(p_l)) \cdot \Pr(\delta^{\omega}_{i,j,k,l} \mid (p_i, p_j, p_k) \in T(p_l)).\end{aligned}$$

Since $\Gamma_1(p_l)$, $\Gamma_2(p_l)$, and $\Gamma_3(p_l)$ are three balls of radius ϵ entirely contained in $\mathcal{U}$, we have

$$\Pr((p_i,p_j,p_k)\in T(p_l)) = \left(\frac{\frac{4}{3}\pi\epsilon^3}{\frac{4}{3}\pi R^3}\right)^3 = \frac{\mu^3\epsilon^9}{n^3}.$$

By Lemmas 3.3 and 4.1, the event $(\delta^{\omega}_{i,j,k,l} \mid (p_i,p_j,p_k)\in T(p_l))$ occurs if and only if the interior of $\mathcal{H}(t_i,t_l,1)\cap\mathcal{U}$ does not contain the center of any ball. Note that the volume of $\mathcal{H}(t_i,t_l,1)\cap\mathcal{U}$ is at most the volume of $\mathcal{H}(t_i,t_l,1)$, which is at most $\frac{4}{3}\pi+\pi(2+e+2\epsilon)$ since the length of t_it_l is at most $e+2+2\epsilon$. It follows that

$$\Pr(\delta^{\omega}_{i,j,k,l} \mid (p_i,p_j,p_k)\in T(p_l)) \geqslant \left(1-\frac{\pi(\frac{4}{3}+2+e+2\epsilon)}{\text{Volume}(\mathcal{U})}\right)^{n-4}.$$

Since $e<2$, we get, after some elementary calculations, that

$$\Pr(\delta^{\omega}_{i,j,k,l} \mid (p_i,p_j,p_k)\in T(p_l)) \geqslant \left(1-\frac{(6+2\epsilon)\mu}{n}\right)^{n-4}. \tag{4.1}$$

We thus have

$$\Pr(\delta^{\omega}_{i,j,k,l}) \geqslant \frac{\mu^3\epsilon^9}{n^3}\left(1-\frac{(6+2\epsilon)\mu}{n}\right)^{n-4}.$$

Since $\left(1-\frac{(6+2\epsilon)\mu}{n}\right)^{n-4}$ tends to $e^{-(6+2\epsilon)\mu}$ when n tends to infinity, we get

$$\Pr(\delta^{\omega}_{i,j,k,l}) = \Omega\left(\frac{1}{n^3}\right). \qquad \square$$

This completes the proof of the lower bound of Theorem 2.1 since the expected number of $T4$-segments amongst n uniformly distributed unit balls is, by Lemmas 3.1 and 4.2,

$$\sum_{(i,j,k,l)\in\mathcal{N}}\sum_{\omega=1}^{12}\Pr(\delta^{\omega}_{i,j,k,l}) = \sum_{(i,j,k,l)\in\mathcal{N}}\sum_{\omega=1}^{12}\Omega\left(\frac{1}{n^3}\right) = \Omega(n).$$

5. The expected size of the visibility complex is linear. In this section we prove Theorem 2.2, that the expected size of the visibility complex of a set of n uniformly distributed unit balls is linear.

We say that the balls are in general position if any k-dimensional face of the visibility complex is a connected set of maximal-free segments tangent to exactly $4-k$ balls. We can assume that the balls are in general position since this occurs with probability 1. We give a bound on the expected number of k-faces for $k=0,\ldots,4$.

LEMMA 5.1. *The expected number of* 0*-faces is* $\Theta(n)$.

Proof. A 0-face of the visibility complex is a maximal-free line segment tangent to four balls. Each maximal-free line segment tangent to four balls contains a $T4$-segment, and each $T4$-segment is contained in one maximal-free line segment. Thus, by Theorem 2.1, the expected number of 0-faces is linear. $\square$

To deal with the faces of dimension $k\geqslant 1$, we divide them into two classes. A k-face is *open* if it is incident to at least one $(k-1)$-face; otherwise, it is *closed*. When the balls are in general position, the number of k-faces incident to a particular

$(k-1)$-face is constant. In the proof of the following lemmas, any constant can be used. However, for completeness, we will use the exact values but without justifying them.

LEMMA 5.2. *The expected number of* 1*-faces is* $\Theta(n)$.

Proof. Note that a 0-face corresponds to a maximal-free segment tangent to four balls, and it is incident to those 1-faces corresponding to free segments tangent to three amongst those four balls. So a 0-face is incident to exactly six 1-faces, which implies that the number of open 1-faces is six times the number of 0-faces, and is thus $\Theta(n)$ by the previous lemma.

Proving that the expected number of closed 1-faces is $O(n)$ can be done in a way very similar to the proof of the upper bound in Theorem 2.1. The difference is that we consider now only three balls, and thus in all proofs we forget ball B_k. We have to consider only $\binom{n}{3}$ triples of balls instead of $\binom{n}{4}$ quadruples, but we remove from the integral the probability $\Pr(p_k \in \mathcal{H}(p_i, p_l, 2) | \mathbf{x}_{i,l} = x) \leqslant \frac{3x+8}{R^3}$. Since $\frac{n}{R^3} = \mu$, this amounts to dividing the terms over which we integrate by $\mu(3x+8)$, which does not change the general shape of the integrals (a polynomial multiplied by an exponential) which are convergent. Notice that B_i, B_j, B_l, and ω now define a set of segments $t_i t_l$, rather than just a single segment. However, those segments define a closed 1-face only if none of them is occluded by one of the $n-3$ remaining balls. Any particular choice of a tangent $t_i t_l$ in the 1-face will give a relevant cylinder $\mathcal{H}(t_i, t_l, 1)$ to use in the proofs. □

LEMMA 5.3. *The expected number of* 2*-faces is* $\Theta(n)$.

Proof. Since a 1-face has five incident 2-faces, the tight linear bound on the number of 1-faces gives a tight linear bound on the number of open 2-faces. The closed case is solved similarly to the proof of the upper bound in Theorem 2.1. We now consider $\binom{n}{2}$ pairs of balls B_i, B_l, and we remove from the integrals the probability $\Pr(p_j, p_k \in \mathcal{H}(p_i, p_l, 2) | \mathbf{x}_{i,l} = x) \leqslant \left(\frac{3x+8}{R^3}\right)^2$, which gives an $O(n)$ bound on the number of closed 2-faces. □

LEMMA 5.4. *The expected numbers of* 3*-faces and* 4*-faces are* $\Theta(n)$.

Proof. A 3-face, corresponding to lines tangent to a ball, can be closed only if $n = 1$. The number of open 3-faces is linear by the fact that in general position a 2-face is incident to four 3-faces. The number of 4-faces is linear since a 3-face is incident to three 4-faces. □

6. Worst-case lower bound. We provide here a $\Omega(n^2)$ lower bound on the number of k-faces in the visibility complex. Recall that for the case of n arbitrarily sized balls, Devillers and Ramos [6] presented a simple $\Omega(n^3)$ lower bound on the number of free segments tangent to four balls, which is also the number of vertices in the visibility complex. Their lower bound (see Figure 6.1) consists of (i) $\frac{n}{3}$ balls such that the view from the origin consists of $\frac{n}{3}$ disjoint disks centered on a circle, (ii) $\frac{n}{3}$ balls such that the view from the origin consists of $\frac{n}{3}$ disks whose boundaries are concentric circles intersecting (in projection) all the disks of (i), and (iii) $\frac{n}{3}$ tiny balls centered around the origin such that from any point on these $\frac{n}{3}$ tiny balls the view of the balls in (i) and (ii) is topologically invariant. Note that finding a $\Omega(n^3)$ lower bound on the number of free segments tangent to four balls, amongst n balls of bounded radii, is to the best of our knowledge open.

PROPOSITION 6.1. *The number of* k*-faces in the visibility complex of* n *disjoint unit balls in* $\mathbb{R}^3$ *is* $\Omega(n^2)$ *for all* k *between* 0 *and* 4.

Proof. We first observe that the size of the visibility complex of n unit balls can trivially be quadratic by having the balls sparsely distributed in the space such that

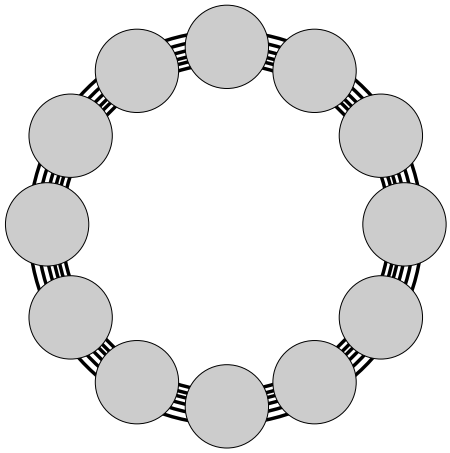


FIG. 6.1. *Quadratic view from the origin* [6].

any pair of balls defines a closed 2-face.

Getting a quadratic number of free lines tangent to four balls amongst a set of n unit balls can be done by taking balls B_i centered at $(2i, 0, 0)$ for $1 \leqslant i \leqslant \frac{n}{2}$ and balls B'_j centered at $(2j, 10, 0)$ for $1 \leqslant j \leqslant \frac{n}{2}$. Then, for any i and j, the line through the points $(2i+1, 0, 1)$ and $(2j+1, 10, 1)$ is free and can be moved down so that it comes into contact with the four balls B_i, B_{i+1}, B'_j, and B'_{j+1}. This argument proves that the number of k-faces, for $0 \leqslant k \leqslant 2$, can be quadratic.

The free segment $(2i, 1, 0)(2j, 9, 0)$ belongs to the 4-face consisting of maximal-free segments with endpoints on B_i and B'_j. Thus there is a quadratic number of 4-faces. The bound also applies to 3-faces by considering lines tangent to B_i and stabbing B'_j.

In the above construction, the balls can be pushed together (they will intersect) so that they fit inside a spherical universe of radius $\sqrt[3]{n/\mu}$ without changing the result. Note also that the above construction can be slightly perturbed to obtain the same result for a set of n unit balls, disjoint or not, with no four centers coplanar. ∎

7. Generalizations. In this section we provide several generalizations of our results.

7.1. Poisson distribution. Consider a set of unit balls whose centers are drawn by a 3D Poisson point process of parameter μ in the universe $\mathcal{U}$. By a *Poisson point process of parameter μ in $\mathcal{U}$* [10], we mean that we generate X random points inside $\mathcal{U}$ so that

$$Pr(X = k) = \frac{(\mu \cdot \text{Volume}(\mathcal{U}))^k \cdot \exp(-\mu \cdot \text{Volume}(\mathcal{U}))}{k!} \tag{7.1}$$

and for any disjoint subsets M and M' of $\mathcal{U}$, the number of the points inside M and the number of points inside M' are independent random variables. Note that (7.1) yields that the expected number of points inside $\mathcal{U}$ is $\mu \cdot \text{Volume}(\mathcal{U}) = \frac{4\pi}{3}n$.

The following simple argument shows that our results extend to this distribution. Let X be the random variable representing the number of centers of unit balls generated by a Poisson point process with parameter μ in $\mathcal{U}$, and let Y be the random variable representing the number of $T4$-segments amongst those balls. The expected

number of $T4$-segments is

$$E(Y) = \sum_{k=0}^{\infty} E(Y|X = k) \cdot \Pr(X = k).$$

Theorem 2.1 gives $E(Y|X = k) = \Theta(k)$ and

$$\Pr(X = k) = \frac{(\frac{4}{3}\pi\ n)^k \cdot \exp(-\frac{4}{3}\pi\ n)}{k!}.$$

Thus

$$\begin{aligned} E(Y) &= \Theta\left(\tfrac{4}{3}\pi\ n\ \exp(-\tfrac{4}{3}\pi\ n)\ \textstyle\sum_{k=1}^{\infty} \frac{(\frac{4}{3}\pi\ n)^{k-1}}{(k-1)!}\right) \\ &= \Theta(n\ \exp(-\tfrac{4}{3}\pi\ n)\ \exp(\tfrac{4}{3}\pi\ n)) = \Theta(n). \end{aligned}$$

Therefore the expected number of $T4$-segments amongst n balls whose centers are generated by a Poisson point process with parameter μ in $\mathcal{U}$ is $\Theta(n)$. Similarly, this bound extends to the expected size of the visibility complex.

We now investigate various models in which we change the shape of the universe or the nature of the objects.

7.2. Smooth convex universe. Our results can be generalized to the case where the universe is no longer a ball but a homothet of a smooth convex set with homothety factor proportional to $\sqrt[3]{n}$. This can be achieved by considering the radius of curvature of the boundary of the universe, instead of R, in the proofs of the lemmas dealing with tangents outside the universe.

7.3. Other objects. Let r_{min} and r_{max} be two strictly positive real constants. In the following, we bound the expected number of $T4$-segments amongst balls whose radii vary in the interval $[r_{min}, r_{max}]$, amongst polyhedra each enclosed between two concentric balls of radii r_{min} and r_{max}, and amongst polygons each enclosed between two concentric circles of radii r_{min} and r_{max}. The centers of the concentric balls or circles are called the centers of the polyhedra or polygons, respectively. In each case a $T4$-segment is called *outer* if the centers of the two extremal objects it is tangent to are farther apart than $6r_{max}$ and are both at distance less than $2r_{max}$ from the boundary of $\mathcal{U}$. Otherwise, the $T4$-segment is called *inner*.

For these models, the proof of the $\Omega(n)$ lower bound on the expected number of $T4$-segments (section 4) generalizes directly because, for the kind of objects we consider, there always exist placements of four of them such that they admit at least one common tangent line with multiplicity one.

7.3.1. Balls of various radii. We have considered a model where all the balls have the same radius. If we allow the radii to vary in the interval $[r_{min}, r_{max}]$, then the proof of the linear upper bound on the expected number of inner $T4$-segments generalizes almost immediately by considering the volumes $\mathcal{H}(p_i, p_l, 2r_{max})$ and $\mathcal{H}(t_i, t_l, r_{min})$ instead of $\mathcal{H}(p_i, p_l, 2)$ and $\mathcal{H}(t_i, t_l, 1)$.

Section 3.2.1 generalizes immediately to prove that the expected number of $T4$-segments tangent to four balls B_i, B_j, B_k, and B_l in that order such that p_i and p_l are closer to one another than $6r_{max}$ is $O(n)$. The only difficult task for extending section 3.2.2 is the proof of the following analogue of Lemma 3.8.

LEMMA 7.1. *When* $\mathbf{x}_{i,l} \geqslant 6r_{max}$, $\mathbf{y}_i \geqslant 2r_{max}$, *and* $\mathcal{L}^{\omega}_{i,j,k,l}$ *occurs, the volume of* $\mathcal{H}(t_i, t_l, r_{min}) \cap \mathcal{U}$ *is greater than* $\frac{\pi}{24} r^2_{min}(\mathbf{x}_{i,l} - 6r_{max})$.

Proof. The proof is similar to the proof of Lemma 3.8. Refer to Figure 7.1. Let m be the midpoint of segment t_it_l and K be the sphere of diameter r_{min} centered on the point c lying on segment t_ip_i at distance $\frac{1}{2}r_{min}$ from t_i. The sphere K is entirely inside $\mathcal{H}(t_i, t_l, r_{min}) \cap \mathcal{U}$, m lies in $\mathcal{H}(t_i, t_l, r_{min})$, and a straightforward computation shows that m is in $\mathcal{U}$ since t_i is in $\mathcal{U}$ at distance at least r_{max} from its boundary and t_l is at distance at most r_{max} from $\mathcal{U}$. Thus $\mathcal{H}(t_i, t_l, r_{min}) \cap \mathcal{U}$ contains the convex hull of K and m, which contains the cone of apex m, of base a disk whose boundary is a great circle of K, and of height the distance from m to the center c of K. Now

$$\begin{aligned}\mathbf{x}_{i,l} = |p_ip_l| &\leqslant |p_ic| + |cm| + |mt_l| + |t_lp_l| \\ &\leqslant r_{max} + |cm| + \frac{1}{2}|t_it_l| + r_{max} \\ &\leqslant 2r_{max} + |cm| + \frac{1}{2}(\mathbf{x}_{i,l} + 2r_{max}).\end{aligned}$$

Thus $|cm| \geqslant \frac{1}{2}\mathbf{x}_{i,l} - 3r_{max}$, and the volume of the cone is at least $\frac{1}{3}\pi(\frac{r_{min}}{2})^2(\frac{1}{2}\mathbf{x}_{i,l} - 3r_{max}) = \frac{\pi}{24}r_{min}^2(\mathbf{x}_{i,l} - 6r_{max})$. □

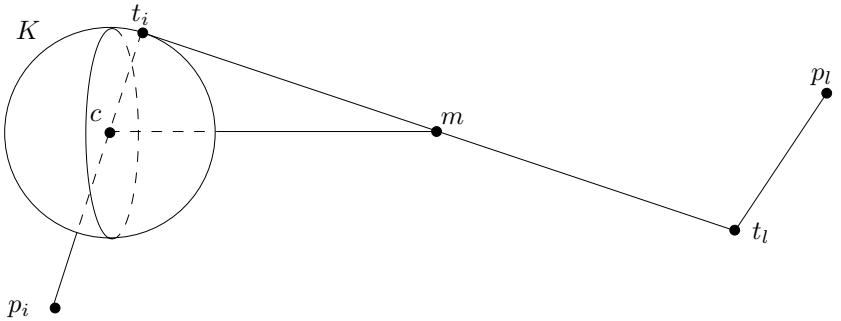


FIG. 7.1. *For the proof of Lemma* 7.1.

The rest of section 3.2.2 generalizes easily for proving that the expected number of $T4$-segments tangent to four balls B_i, B_j, B_k, and B_l in that order such that p_i and p_l are farther apart than $6r_{max}$ and p_i is farther than $2r_{max}$ from the boundary of $\mathcal{U}$ is $O(n)$. Hence the expected number of inner $T4$-segments is $O(n)$.

Our proof cannot be extended to provide a linear upper bound on the expected number of outer $T4$-segments. This is because, if balls B_i and B_l are of radius r_{max}, then a line segment t_it_l tangent to B_i and B_l might be outside $\mathcal{U}$ and at distance greater than r_{min} from its boundary. Then $\mathcal{H}(t_i, t_l, r_{min})$ does not intersect $\mathcal{U}$, and we cannot bound $\mathcal{H}(t_i, t_l, r_{min}) \cap \mathcal{U}$ from below by a positive constant as in Lemma 3.12, which is crucial for the proof of Lemma 3.13 and thus for Proposition 3.18.

However, by not taking into account the occlusion in the proof of Proposition 3.18, we get that the expected number of outer $T4$-segments is $O(n^2)$. Refer to the proof of Proposition 3.18, and consider $I_{x>6r_{max}, y<2r_{max}}$, the analogue of $I_{x>6,y<1}$ for this case. The analogues of Lemmas 3.4 and 3.5 yield that

$$\Pr(\delta^{\omega}_{i,j,k,l} \mid \mathbf{x}_{i,l} = x,\ \mathbf{y}_i = y,\ \mathbf{y}_l \leqslant \mathbf{y}_i) \leqslant \frac{(3\,x\,r_{max}^2 + 8\,r_{max}^3)^2}{R^6}.$$

Lemma 3.16 still holds, and we can easily prove the analogue of Lemma 3.17. Both

results imply that

$$I_{x>6r_{max},y<2r_{max}} \leqslant \int_{x=6r_{max}}^{2R} \int_{y=0}^{2r_{max}} \frac{(3\,x\,r_{max}^2 + 8\,r_{max}^3)^2}{R^6} \cdot \frac{6\,x\,y\,dx}{R^3} \cdot \frac{3\,dy}{R}$$
$$\in O\left(\frac{1}{R^6}\right) = O\left(\frac{1}{n^2}\right).$$

Hence the expected number of inner $T4$-segments is $O(n)$, and the expected number of outer $T4$-segments is $O(n^2)$. This still improves the result of Durand, Drettakis, and Puech [9], who proved a bound of $O(n^{8/3})$ for the same model.

In this section we have assumed that the sphere centers are uniformly distributed, but we have made no assumption on the distribution of the radii of the spheres in the interval $[r_{min}, r_{max}]$, which are thus assumed to be worst-case. The addition of some hypothesis on the radii distribution may yield better results on the number of outer $T4$-segments.

7.3.2. Polyhedra of bounded aspect ratio. Consider polyhedra of constant complexity, each enclosed between two concentric balls of radii r_{min} and r_{max} whose centers are uniformly distributed in $\mathcal{U}$. In such a case, as for balls of various radii, the $O(n)$ bound on the expected number of inner $T4$-segments immediately applies, as well as the $O(n^2)$ bound on the expected number of outer $T4$-segments.

7.3.3. Polygons of bounded aspect ratio. Our proof technique can also be generalized to nonfat 3D objects such as polygons. Consider polygons of constant complexity enclosed between two coplanar concentric circles of radii r_{min} and r_{max} and whose centers and normals are independently chosen from the uniform distributions over $\mathbb{R}^3$ and $\mathbb{S}^2$. Let $T_1, \ldots, T_n$ be such polygons with respective normals $\mathbf{n}_1, \ldots, \mathbf{n}_n$ and centers $p_1, \ldots, p_n$.

Four polygons T_i, T_j, T_k, and T_l have a common tangent line that meet them in that order only if p_j and p_k lie in $\mathcal{H}(p_i, p_l, 2r_{max})$. This implies, as in section 3.2.1, that the expected number of $T4$-segments tangent to four polygons T_i, T_j, T_k, and T_l in that order such that p_i and p_l are closer to one another than some constant, say $6r_{max}$, is $O(n)$.

When such a tangent, denoted t_it_l, exists, it is not occluded only if, for any $\gamma \neq i, j, k, l$, point p_γ does not lie in the interior of $\mathcal{H}(t_i, t_l, r_{min}\cos\theta_\gamma)$, where θ_γ denotes the angle between $\mathbf{n}_\gamma$ and the supporting line of t_it_l (see Figure 7.2(a) and Lemma 3.3). Let γ be an integer distinct from i, j, k, and l. By the total probability theorem, the probability that T_γ does not occlude the tangent line segment t_it_l is bounded from above by

$$\int_{\theta=0}^{\pi/2} \Pr\left(p_\gamma \notin \mathcal{H}(t_i, t_l, r_{min}\cos\theta_\gamma) \mid \theta_\gamma = \theta\right) \cdot \Pr(\theta \leqslant \theta_\gamma < \theta + d\theta).$$

Similarly as in Lemma 7.1, when the tangent t_it_l exists, $\mathbf{x}_{i,l} \geqslant 6r_{max}$ and $\mathbf{y}_i \geqslant 2r_{max}$, and the volume of $\mathcal{H}(t_i, t_l, r_{min}) \cap \mathcal{U}$ is greater than $\frac{\pi}{24}(r_{min}\cos\theta_\gamma)^2(\mathbf{x}_{i,l} - 6r_{max})$. Thus

$$\Pr\left(p_\gamma \notin \mathcal{H}(t_i, t_l, r_{min}\cos\theta_\gamma) \mid \theta_\gamma = \theta\right) \leqslant 1 - \frac{(r_{min}\cos\theta_\gamma)^2(\mathbf{x}_{i,l} - 6r_{max})}{32R^3}.$$

The probability that θ_γ is in between θ and $\theta + d\theta$ is $\sin\theta\,d\theta$, which corresponds to twice the area of the spherical shell between the latitudes θ and $\theta + d\theta$ on the unit

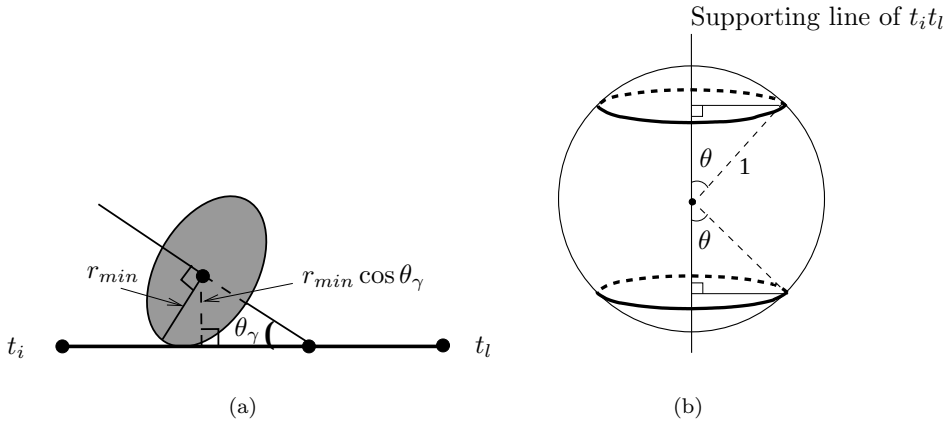


FIG. 7.2. *Illustration for the case of polygons of bounded aspect ratio.*

sphere, divided by the area of the unit sphere (see Figure 7.2(b)). Thus when p_i is at distance greater than $6r_{max}$ from p_l and at distance greater than $2r_{max}$ from the boundary of $\mathcal{U}$, the probability that T_γ does not occlude the tangent line segment $t_i t_l$ is bounded from above by

$$\int_{\theta=0}^{\pi/2} \left(1 - \frac{(r_{min}\cos\theta_\gamma)^2(\mathbf{x}_{i,l} - 6r_{max})}{32R^3}\right) \sin\theta \, d\theta = 1 - \frac{r_{min}^2\,(\mathbf{x}_{i,l} - 6r_{max})}{96\,R^3}.$$

Then, similarly as in Lemma 3.9, the probability that the tangent line segment $t_i t_l$ is not occluded, when p_i is at distance greater than $6r_{max}$ from p_l and at distance greater than $2r_{max}$ from the boundary of $\mathcal{U}$, is at most

$$55\exp\left(-\frac{\mu\, r_{min}^2\,(\mathbf{x}_{i,l} - 6r_{max})}{96}\right).$$

We thus get the analogue of Proposition 3.10 for the model considered here, which implies that the expected number of $T4$-segments tangent to four polygons T_i, T_j, T_k, and T_l in that order such that p_i and p_l are farther apart than $6r_{max}$ and p_i is farther than $2r_{max}$ from the boundary of $\mathcal{U}$ is $O(n)$.

We thus get that the expected number of inner $T4$-segments is $O(n)$. Moreover, as for balls of various radii, the expected number of outer $T4$-segments is $O(n^2)$.

8. Conclusion. In this paper, we proved that the expected number of $T4$-segments amongst n uniformly distributed unit balls in $\mathbb{R}^3$ is $\Theta(n)$. We also proved that the expected size of the visibility complex of n uniformly distributed unit balls is $\Theta(n)$. Equivalently, the expected number of combinatorially different visibility events amongst n uniformly distributed unit balls is $\Theta(n)$. We then proved that $\Theta(n)$ also bounds the expected number of $T4$-segments occurring not too close to the boundary of the universe for various other models such as n uniformly distributed polyhedra, or polygons, of bounded aspect ratio and constant complexity. For these models, we also provided a $O(n^2)$ bound on the expected number of all the $T4$-segments.

This paper is an attempt to analyze the average-case behavior of the size of visibility structures. The distribution models of scene objects investigated here are theoretical in nature since objects in graphics scenes are seldom distributed uniformly

or by a Poisson process. However, our results are important in a context where there are few rigorous results, either theoretical or experimental. They provide theoretical ground to support the empirical evidence indicating that the worst-case upper bound on the number of visibility events is largely pessimistic in practical situations. As a consequence, there is reason to believe that an output-sensitive algorithm for computing all visibility events may work in practice.

Practitioners will be concerned about the size of the constant hidden in the Θ notation. We have calculated (in the proofs of section 3) this constant to be no larger than $2^{16}\,\mu^3 + 2^{31}\,\mu + 2^{37}\,e^{-\mu/3}\,(\mu^2 + 1/\mu^2)$. Of course this is shocking. We suppose that the constant is actually much smaller. However, estimating it in practice is a difficult problem which is still to be solved. After solving this problem, an interesting experiment will be to compare the number of visibility events occurring in a realistic graphic scene with the theoretical bound for uniformly distributed objects.

The results proved here also provide new insight on the complexity of other visibility structures. Consider, for instance, the aspect graph, a partition of viewpoint space into maximal connected regions by surfaces along which visibility events are observed. As explained in [19], the complexity of the aspect graph is dominated by δ^m, where δ is the degree of the surface corresponding to lines "tangent" to three objects and m the dimension of the viewpoint space. For a scene composed of n disjoint spheres, δ is trivially $O(n^3)$, so the aspect graph has $O(n^6)$ orthographic views and $O(n^9)$ perspective views. However, the results of this paper show that the expected value of δ is $\Theta(n)$ since the expected number of families of lines tangent to three objects (related to the 1-faces of the visibility complex) is linear and the degree of each family is bounded. It would thus be interesting to get a good bound on the expected value of δ^2 and δ^3, which is related to bounding the expected value of the square and the cube of the number of combinatorially different visibility events. Note that the former would also give the standard deviation of the expected number of combinatorially different visibility events. Similar observations hold for the polyhedral case.

Appendix A. Volume of the intersection of a 3D hippodrome with a ball. Recall that $\mathcal{U}$ is a ball of radius R centered at O. Let B_i and B_l be two unit balls whose centers p_i and p_l are in $\mathcal{U}$, within distance 1 from its boundary, and distance $x \geqslant 6$ apart. Let t_it_l be a line segment tangent to B_i and B_l at its endpoints. The section is devoted to the proof of the following proposition, which leads directly to Lemma 3.12.

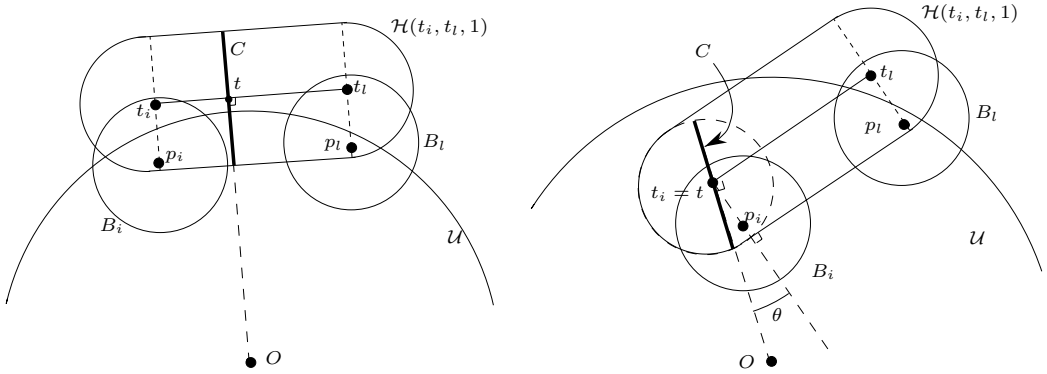
PROPOSITION A.1. *For any $0 \leqslant s \leqslant 1$ such that segment t_it_l is at distance less than $R+1-s$ from O, the volume of $\mathcal{H}(t_i, t_l, 1) \cap \mathcal{U}$ is larger than $\frac{1}{6\sqrt{2}}\,(x-5)\,s\,\sqrt{s}$.*

We proceed as follows. Let v be such that the distance from O to the segment t_it_l is $R+1-v$, and let t be the point on segment t_it_l closest to O (see Figure A.1). Assume without loss of generality that t is closer to t_i than to t_l. Let C (resp., D) be the unit radius circle (resp., disk) centered at t in the plane, denoted $\mathcal{P}$, containing the vectors $\overrightarrow{Ot}$ and the cross-product of $\overrightarrow{Ot}$ and $\overrightarrow{t_it_l}$. Let θ be the angle between the plane orthogonal to t_it_l and $\mathcal{P}$. We first prove the following lemma.

LEMMA A.2. *The volume of $\mathcal{H}(t_i, t_l, 1) \cap \mathcal{U}$ is greater than*

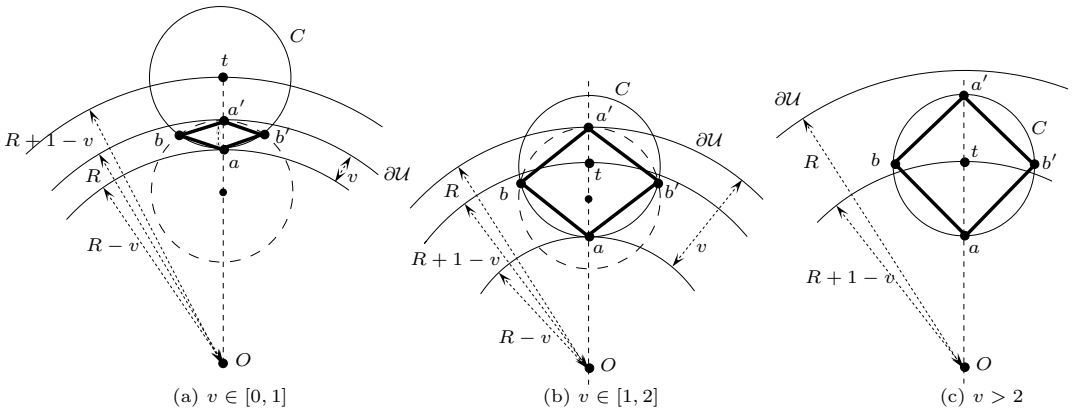
$$\frac{1}{3}\min\left(2, \frac{v\sqrt{v}}{\sqrt{2}}\right) \cdot \min\left(\frac{x-2}{2}, (x-2)\cos\theta - 1\right).$$

Proof. Let a denote the closest point on C from O, a' the farthest point in $D \cap \mathcal{U}$



(a) t is not equal to t_i or t_l; $\theta = 0$. (b) t is equal to t_i.

FIG. A.1. *For the definition of t and C (C is shown from the side view).*



(a) $v \in [0, 1]$ (b) $v \in [1, 2]$ (c) $v > 2$

FIG. A.2. *For the definition of a, a', b, b'.*

from O, and b and b' the two points of intersection of C and the perpendicular bisector of segment aa' (see Figure A.2).

The volume of $\mathcal{H}(t_i, t_l, 1) \cap \mathcal{U}$ is greater than the volume of the convex hull of a, b, a', b', and p_l because $\mathcal{H}(t_i, t_l, 1) \cap \mathcal{U}$ is convex and contains these five points. The volume of this polyhedron is equal to one third of the area of its base, the quadrilateral with vertices a, b, a', b' times its height, the distance from p_l to the plane $\mathcal{P}$ containing a, b, a', b'.

We first compute a lower bound on the area of the quadrilateral with vertices a, b, a', b'. If $v \leqslant 2$ (see Figure A.2(a) and (b)), then the length of aa' is equal to v, and a simple calculation gives that the length of bb' is equal to $2\sqrt{v - \frac{v^2}{4}} \geqslant \sqrt{2v}$. Thus the area of the quadrilateral a, b, a', b' is greater than $\frac{v\sqrt{v}}{\sqrt{2}}$. If $v > 2$ (see Figure A.2(c)), then C is entirely contained in $\mathcal{U}$, and the area of the quadrilateral a, b, a', b' is equal to 2. Thus, the area of the quadrilateral is at least $\min(2, \frac{v\sqrt{v}}{\sqrt{2}})$.

The volume of the polyhedron is thus greater than $\frac{1}{3}\min(2, \frac{v\sqrt{v}}{\sqrt{2}})$ times the distance from p_l to the plane $\mathcal{P}$. We consider two cases.

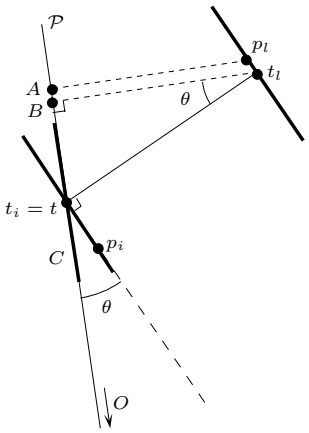


FIG. A.3. *The height from p_l to $\mathcal{P}$ is greater than $|t_i t_l| \cos\theta - 1$.*

First, suppose that t belongs to the interior of the segment $t_i t_l$ (see Figure A.1(a)). Then the height is equal to the distance from t_l to t since p_l and a, b, a', b' belong, respectively, to the two planes, orthogonal to $t_i t_l$ and passing through t_l and t, respectively. Since t_i and t_l belong to B_i and B_l, they are at least distance $x-2$ apart; thus t and t_l are at least distance $\frac{x-2}{2}$ apart. Thus, the height from p_l to $\mathcal{P}$ is at least $\frac{x-2}{2}$.

Second, suppose that $t = t_i$ (see Figure A.1(b)); $t \neq t_l$ since we assumed that t is closer to t_i than to t_l. Refer to Figure A.3. Let A and B be the orthogonal projections of p_l and t_l onto $\mathcal{P}$, respectively. Note that the lengths of Ap_l and Bt_l are the distances from p_l and t_l to the plane $\mathcal{P}$, respectively.

Considering the triangle $\triangle Ap_l t_l$ and that the distance between t_l and p_l is 1, we obtain that $|Ap_l| \geqslant |At_l| - |t_l p_l| = |At_l| - 1$, where $|ab|$ denotes the length of segment ab. Since $A \in \mathcal{P}$ and the length of Bt_l is the distance from t_l to the plane $\mathcal{P}$, the length of At_l is greater than that of Bt_l; thus $|Ap_l| \geqslant |Bt_l| - 1$.

To bound the length of Bt_l, we now consider the triangle $\triangle Bt_l t$. The angle $\angle Bt_l t$ is the angle between the normal of the plane $\mathcal{P}$ and $t_i t_l$, that is, by definition, θ. So the length of Bt_l is the length of $t_i t_l$ times $\cos\theta$, and, since $|t_i t_l|$ is at least $x-2$, $|Bt_l|$ is greater than $(x-2)\cos\theta$. Thus the length of Ap_l is greater than $(x-2)\cos\theta - 1$.

Hence the distance from p_l to the plane $\mathcal{P}$ is greater than $\min(\frac{x-2}{2}, (x-2)\cos\theta - 1)$, and thus the volume of $\mathcal{H}(t_i, t_l, 1) \cap \mathcal{U}$ is greater than $\frac{1}{3}\min(2, \frac{v\sqrt{v}}{\sqrt{2}}) \cdot \min(\frac{x-2}{2}, (x-2)\cos\theta - 1)$. ∎

The following lemma bounds $\cos\theta$.

LEMMA A.3. *The angle θ is such that $\cos\theta \geqslant \frac{\sqrt{7}}{4}$.*

Proof. Note first that this lemma is intuitively obvious. Indeed (see Figure A.1(b)), if x is sufficiently large and if t_i is the closest point on segment $t_i t_l$ to O, then the angle between the plane supporting C and the segment $t_i t_l$ is necessarily close to $\pi/2$, which implies that θ is close to 0. We now prove the lemma.

Refer to Figure A.3, and consider the triangle $\triangle Ot_i t_l$. Let $|ab|$ denote the length of segment ab. Then the law of cosines yields

$$\begin{aligned} |Ot_l|^2 &= |Ot_i|^2 + |t_i t_l|^2 - 2 \cdot |Ot_i| \cdot |t_i t_l| \cdot \cos(\tfrac{\pi}{2} + \theta) \\ &= |Ot_i|^2 + |t_i t_l|^2 + 2 \cdot |Ot_i| \cdot |t_i t_l| \cdot \sin\theta, \end{aligned}$$

which gives that

$$\sin\theta = \frac{|Ot_l|^2 - |Ot_i|^2 - |t_it_l|^2}{2 \cdot |Ot_i| \cdot |t_it_l|}.$$

The centers p_i and p_l of balls B_i and B_l are distance $x \geqslant 6$ apart and at distance less than 1 from the boundary of $\mathcal{U}$, so $|t_it_l| \geqslant 4$, $|Ot_i| \geqslant R-2$ and $|Ot_l| \leqslant R+1$. Hence

$$\sin\theta \leqslant \frac{(R+1)^2 - (R-2)^2 - 4^2}{2 \cdot (R-2) \cdot 4} \leqslant \frac{6(R-2)}{8(R-2)} = \frac{3}{4}.$$

Using $\cos\theta = \sqrt{1-(\sin\theta)^2}$ proves that $\cos\theta \geqslant \frac{\sqrt{7}}{4}$. □

We can now conclude the proof of Proposition A.1. For any $0 \leqslant s \leqslant 1$, if segment t_it_l is at distance $R+1-v \leqslant R+1-s$ from the center of $\mathcal{U}$, then $v \geqslant s$. By Lemma A.3, $(x-2)\cos\theta - 1 \geqslant \frac{x-5}{2}$, which means that $\min(\frac{x-2}{2}, (x-2)\cos\theta - 1) \geqslant \frac{x-5}{2}$. Thus Lemma A.2 gives that the volume of $\mathcal{H}(t_i,t_l,1) \cap \mathcal{U}$ is greater than $\frac{1}{6\sqrt{2}}(x-5)\min(2\sqrt{2}, v\sqrt{v}) \geqslant \frac{1}{6\sqrt{2}}(x-5)\min(2\sqrt{2}, s\sqrt{s}) = \frac{1}{6\sqrt{2}}\, s\sqrt{s}\,(x-5)$ since $s \leqslant 1$. Hence the volume of $\mathcal{H}(t_i,t_l,1) \cap \mathcal{U}$ is greater than $\frac{1}{6\sqrt{2}}\, s\sqrt{s}\,(x-5)$.

Appendix B. Volume of K. Recall that $\mathcal{U}$ is a ball of radius R centered at O, and let p_i and p_l be two points in $\mathcal{U}$ within distance 1 of its boundary and distance x apart. Let y be a real number such that $0 \leqslant y < 1$. Let F be the open ball with center O and radius $R+1-y^{\frac{2}{3}}$ and ∂F its frontier. Let E be the part of $\mathcal{H}(p_i,p_l,1)$ that is outside F, and let K be the intersection of $\mathcal{U}$ with the union of all unit balls centered on points in E (see Figure B.1). This section is devoted to the proof of the following proposition used in the proof of Lemma 3.15.

PROPOSITION B.1. *The volume of K is bounded from above by $12\,\pi^2\,(x+6)\,y$.*

LEMMA B.2. *If $z \in \mathcal{U}$ is at distance less than 1 from E, then z is at distance less than 1 from $E \cap \partial F$.*

Proof. Let $z \in \mathcal{U}$ and $w \in E$ be two points at distance less than 1, and refer to Figure B.1. Let w' be the point of intersection of ∂F and the ray from O through w. For any ball B centered in $\mathcal{U}$, $B \setminus F$ lies in the cone of center O and base $B \cap \partial F$. Thus $E = \mathcal{H}(p_i,p_l,1) \setminus F$ lies in the cone of center O and base $E \cap \partial F$. Hence the ray from O through w lies in this cone and $w' \in E \cap \partial F$. On the other hand, $|zw'| \leqslant |zw|$ since $z \in F$, $w' \in \partial F$, and w lies outside F on the ray from O through w'. Thus, since $w' \in E \cap \partial F$ and $|zw| < 1$ by hypothesis, the distance from z to $E \cap \partial F$ is less than 1. □

The above lemma implies that K is the intersection of $\mathcal{U}$ with the union of all unit balls centered on $E \cap \partial F$. To bound the volume of K, we enclose $E \cap \partial F$ in a subset of ∂F that will be easier to deal with.

Let $B(p)$ denote the ball of unit radius centered at p. Let $\pi(p)$ be the point that maximizes (under inclusion) the intersection $\partial F \cap B(q)$ for all q on the ray from O through p. A simple computation yields that the distance between $\pi(p)$ and O is

$$R_y = \sqrt{(R+1-y^{\frac{2}{3}})^2 - 1}.$$

Thus π is the orthogonal projection onto the sphere centered at O of radius R_y. Now let $\pi'(p)$ be the point that maximizes (under inclusion) the intersection $\partial F \cap B(q)$ for all q on the radius of $\mathcal{U}$ through p (that is the part inside $\mathcal{U}$ of the ray from O through p). Similarly, π' is the orthogonal projection onto the sphere centered at O of radius

$$R' = \min(R, R_y).$$

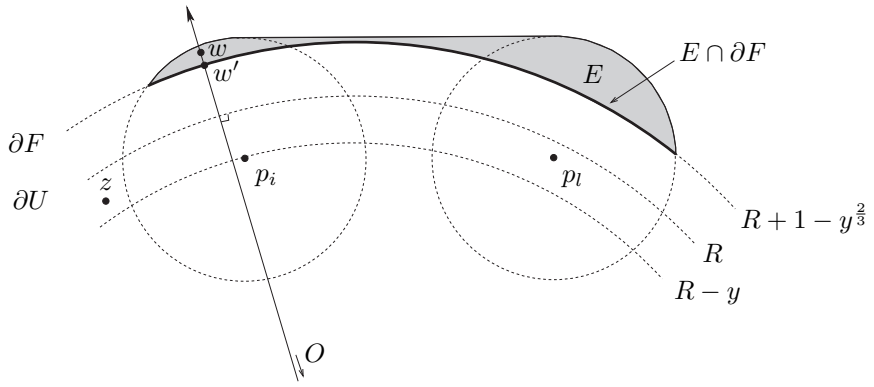


FIG. B.1. *The part E of $\mathcal{H}(p_i, p_l, 1)$ outside F.*

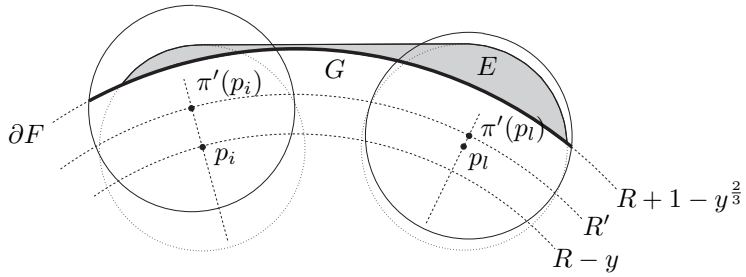


FIG. B.2. *G, a part of ∂F enclosing $E \cap \partial F$.*

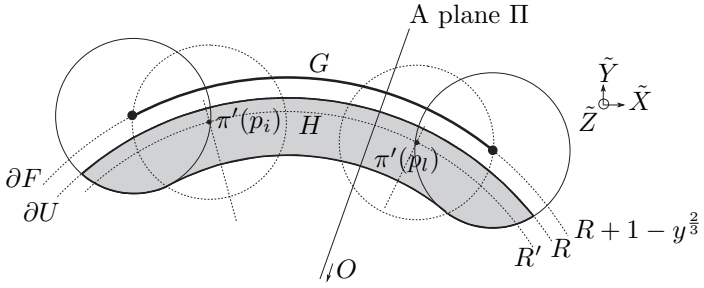


FIG. B.3. *The region H and a plane* Π.

Let G be the union of the spherical caps $\partial F \cap B(\pi'(p))$ for all p on the segment from p_i to p_l (see Figure B.2). Let H denote the points of $\mathcal{U}$ at distance less than or equal to 1 from G (see Figure B.3).

LEMMA B.3. $K \subseteq H$.

Proof. $E \cap \partial F$ is the union of $\partial F \cap B(p)$ for all p on the segment $p_i p_l$. Furthermore, for any such p, $\partial F \cap B(p) \subseteq \partial F \cap B(\pi'(p))$ by definition of π' since $p \in \mathcal{U}$. Thus $E \cap \partial F$ is contained in G.

By Lemma B.2, K is the intersection of $\mathcal{U}$ with the union of all unit balls centered on $E \cap \partial F$. Thus K is contained in H, with the union of all unit balls centered in G. ∎

To bound the volume of H from above, we first bound the area of its section by planes Π that contain O and are orthogonal to the plane, denoted (O, p_i, p_l),

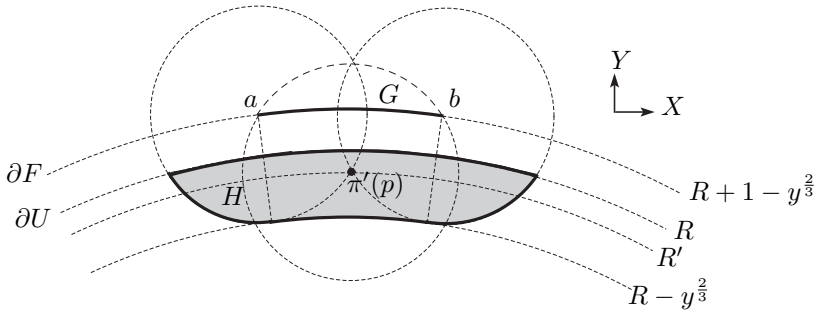


FIG. B.4. *Section of H by a plane Π intersecting segment p_ip_l at p.*

containing O, p_i, and p_l (see Figures B.3 and B.4).

LEMMA B.4. *The area of $\Pi \cap H$ is less than $12\,\pi\, y$.*

Proof. The section of G by a plane Π is a circular arc on ∂F. If Π intersects the segment p_ip_l, let p denote the point of intersection; then the circular arc is the intersection of ∂F and the disk $B(\pi'(p))\cap\Pi$ (refer to Figures B.2 and B.4). Otherwise, the circular arc is the intersection of ∂F and the disk $B(\pi'(p_i)) \cap \Pi$ or $B(\pi'(p_l)) \cap \Pi$ (see Figure B.2). The disk has radius 1 in the former case and radius less than one in the latter case. In both cases the center of the disk is at distance R' from O. Thus the length of the circular arc $G\cap\Pi$ is maximal if and only if Π intersects the segment p_ip_l. Thus the area of $\Pi\cap H$ is maximal if and only if Π intersects the segment p_ip_l. Hence we can assume that Π is such a plane. Let p denote its intersection with segment p_ip_l.

Let a and b denote the endpoints of $G\cap\Pi$, and refer to Figure B.4. Points a and b are the intersection of ∂F and the circle in Π of radius 1 centered at $\pi'(p)$. The lines (Oa) and (Ob) split $\Pi\cap H$ into three parts: a left, a central, and a right part. Symmetries with respect to the lines (Oa) and (Ob) send the left and right parts into the central one. Hence, the area of $\Pi \cap H$ is bounded by three times the area of its central part. This part is delimited by the two rays from O through a and b, and the two circles in Π with center O and radii R and $R - y^{\frac{2}{3}}$. So, if α denotes the length of the circular arc ab, the area A of the central part is

$$A = \frac{\alpha}{2\pi(R+1-y^{\frac{2}{3}})} \cdot \pi(R^2 - (R-y^{\frac{2}{3}})^2) = \alpha\,\frac{2Ry^{\frac{2}{3}} - y^{\frac{4}{3}}}{2(R+1-y^{\frac{2}{3}})} \leqslant \alpha\, y^{\frac{2}{3}}.$$

We now bound the length α of the arc ab. We choose an orthonormal frame $(\pi'(p), X, Y)$ in Π such that O has coordinates $(0, -R')$ (see Figure B.4). Recall that a is one of the intersection points of the circle centered at $\pi'(p)$ of radius 1 and the circle centered at O of radius $R + 1 - y^{\frac{2}{3}}$. A simple computation yields that the coordinates (X_a, Y_a) of a are equal to

$$Y_a = \frac{(R+1-y^{\frac{2}{3}})^2 - 1 - R'^2}{2R'}, \qquad |X_a| = \sqrt{1 - Y_a^2}.$$

If $R' = R$, then

$$Y_a = \frac{y^{\frac{4}{3}} + 2R - 2Ry^{\frac{2}{3}} - 2y^{\frac{2}{3}}}{2R} = 1 - y^{\frac{2}{3}}\left(1 + \frac{2 - y^{\frac{2}{3}}}{2R}\right) \geqslant 1 - 2y^{\frac{2}{3}},$$

which implies that

$$|X_a| \leqslant \sqrt{1 - (1-2y^{\frac{2}{3}})^2} = \sqrt{4y^{\frac{2}{3}} - 4y^{\frac{4}{3}}} \leqslant 2y^{\frac{1}{3}}.$$

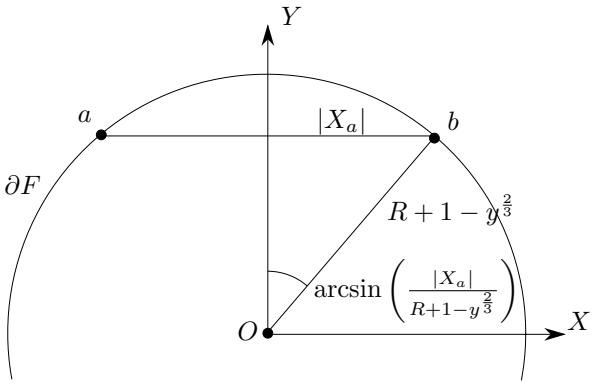


FIG. B.5. *The length of the circular arcs ab.*

Now, if $R' \neq R$, then $(R+1-y^{\frac{2}{3}})^2 - 1 \leqslant R^2$ by definition. Expanding this inequality yields

$$y^{\frac{4}{3}} + 2R - 2Ry^{\frac{2}{3}} - 2y^{\frac{2}{3}} \leqslant 0,$$

$$y^{\frac{2}{3}} \geqslant \frac{y^{\frac{4}{3}} + 2R}{2(R+1)} \geqslant \frac{R}{R+1} \geqslant \frac{1}{2}.$$

Thus $\sqrt{2y^{\frac{2}{3}}} \geqslant 1$, and since $|X_a| = \sqrt{1-Y_a^2} \leqslant 1$ we get $|X_a| \leqslant \sqrt{2}y^{\frac{1}{3}}$. Hence, in both cases,

$$|X_a| \leqslant 2y^{\frac{1}{3}}.$$

Thus the length of the circular arc ab is (see Figure B.5)

$$\alpha = (R+1-y^{\frac{2}{3}}) \cdot 2\arcsin\left(\frac{|X_a|}{R+1-y^{\frac{2}{3}}}\right) \leqslant (R+1-y^{\frac{2}{3}}) \cdot 2\arcsin\left(\frac{2y^{\frac{1}{3}}}{R+1-y^{\frac{2}{3}}}\right).$$

A straightforward computation shows that $\arcsin(x) - \pi\, x \leqslant 0$ for any $x \in [0,1]$. Thus

$$\alpha \leqslant (R+1-y^{\frac{2}{3}}) \cdot 2\,\pi \frac{2y^{\frac{1}{3}}}{R+1-y^{\frac{2}{3}}} = 4\,\pi\, y^{\frac{1}{3}}.$$

Since the area A of the middle part is less than or equal to $\alpha\, y^{\frac{2}{3}}$,

$$A \leqslant 4\,\pi\, y^{\frac{1}{3}}\, y^{\frac{2}{3}} = 4\,\pi\, y.$$

This implies that the area of $\Pi \cap H$ is less than or equal to $12\,\pi\, y$. □

LEMMA B.5. *The volume of H is bounded from above by $12\,\pi^2\,(x+6)\,y$.*

Proof. We express the volume of H by an integral using spherical coordinates (r,θ,ϕ) in an orthogonal frame $(O,\tilde{X},\tilde{Y},\tilde{Z})$ such that the plane $(O,\tilde{X},\tilde{Y})$ contains p_i and p_l (see Figure B.3). A plane $\theta =$ constant contains the $\tilde{Z}$-axis and thus is a plane Π. Let $1_H(r,\theta,\phi)$ denote the indicator function of H; $1_H(r,\theta,\phi)$ is equal to 1 if the point of coordinates (r,θ,ϕ) belongs to H and to 0 otherwise. Then

$$\text{Volume of } H = \int_\phi \int_r \int_\theta 1_H(r,\theta,\phi) \cdot r^2 \sin\phi\, dr\, d\theta\, d\phi.$$

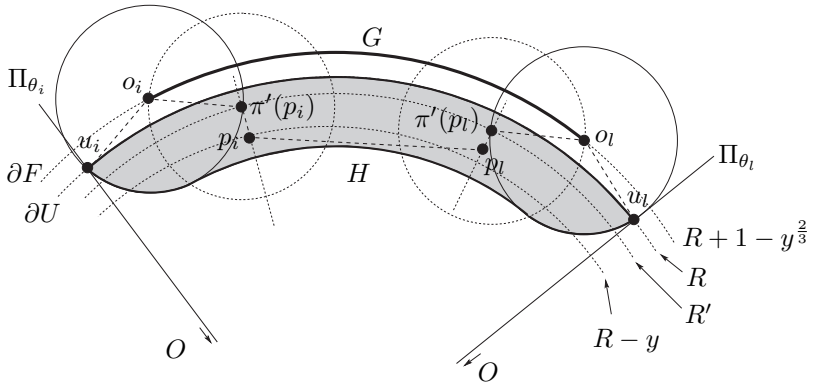


FIG. B.6. *For computing a bound on* $\Delta\theta$.

Since H is inside $\mathcal{U}$, $r \cdot 1_H(r,\theta,\phi) \leqslant R \cdot 1_H(r,\theta,\phi)$. Moreover, $\sin\phi \leqslant 1$; thus

$$\text{Volume of } H \leqslant R \int_\theta \left(\int_\phi \int_r 1_H(r,\theta,\phi) \cdot r \, dr \, d\phi \right) d\theta.$$

The double integral in parentheses is equal to the area of the section of H by a plane $\Pi_\theta : \theta = \text{constant}$. By Lemma B.4, this area is less than $12\,\pi\,y$, which is independent of θ. Moreover, the area is equal to 0 when Π_θ does not intersect H. Let $\Delta\theta$ denote the angle between the two extreme planes Π_θ that intersect H. Thus we have

$$\text{Volume of } H \leqslant R \cdot 12\,\pi\,y \cdot \Delta\theta.$$

We now bound $\Delta\theta$. Refer to Figure B.6. Let Π_{θ_i} and Π_{θ_l} be the two extreme planes that intersect H. Let u_i and u_l be the two points of intersection of H with Π_{θ_i} and Π_{θ_l}, respectively; u_i and u_l lie on $\partial\mathcal{U}$. Let o_i and o_l be the two points in G at distance 1 from u_i and u_l, respectively. $\pi'(p_i)$ and $\pi'(p_l)$ are at distance 1 from o_i and o_l, respectively.

The angle between the two extreme planes Π_{θ_i} and Π_{θ_l} is, as before,

$$\Delta\theta = 2\arcsin\frac{|u_iu_l|/2}{R} \leqslant 2\,\pi\frac{|u_iu_l|/2}{R} = \pi\frac{|u_iu_l|}{R}.$$

Now we bound $|u_iu_l|$ by the length of the polygonal line shown in Figure B.6.

$$\begin{aligned}|u_iu_l| &\leqslant |u_io_i| + |o_i\pi'(p_i)| + |\pi'(p_i)p_i| + |p_ip_l| + |p_l\pi'(p_l)| + |\pi'(p_l)o_l| + |o_lu_l| \\ &= 1 + 1 + |\pi'(p_i)p_i| + x + |p_l\pi'(p_l)| + 1 + 1.\end{aligned}$$

We show that $|\pi'(p_i)p_i|$ and $|p_l\pi'(p_l)|$ are less than 1. $\pi'(p_i)$ is inside $\mathcal{U}$ at distance less than 1 from ∂F, which lies outside $\mathcal{U}$. Thus $\pi'(p_i)$ is inside $\mathcal{U}$ at distance less than 1 from its frontier. Point p_i is also inside $\mathcal{U}$ at distance less than 1 from its frontier. Since p_i and $\pi'(p_i)$ are on the same ray starting from O, they are at distance less than 1 apart and similarly for $\pi'(p_l)$ and p_l. Hence

$$\Delta\theta \leqslant \pi\frac{|u_iu_l|}{R} \leqslant \pi\frac{x+6}{R}.$$

Therefore

$$\text{Volume of } H \leqslant R \cdot 12\,\pi\,y \cdot \Delta\theta \leqslant 12\,\pi^2\,y\,(x+6). \qquad \square$$

Proposition B.1 follows from Lemmas B.3 and B.5.

Appendix C. Volume of the intersection of two spherical shells.

We prove in this section the following proposition used in the proof of Lemma 3.17.

PROPOSITION C.1. *Let $R > 0$, $x \in [6, 2R]$, $y \in [0, 1]$, and let p be a point at distance $R - y$ from O. The volume of the intersection of the region in between the two spheres centered at p and of radii x and $x + dx$, and the region in between the two spheres centered at O and of radii R and $R - y$ (see Figure* C.1*), is bounded from above by $8\,\pi\,x\,y\,dx$.*

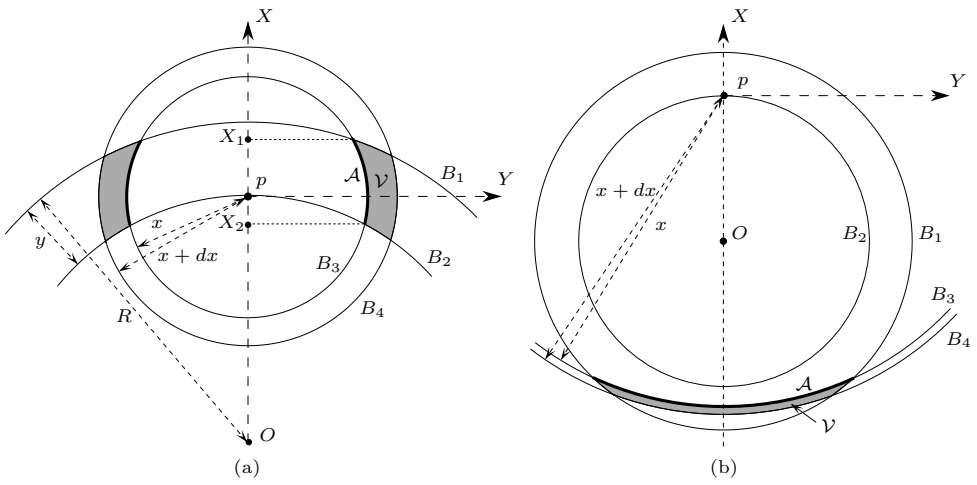


FIG. C.1. *For the proof of Proposition* C.1.

Proof. Define the balls B_1 with center O and radius R, B_2 with center O and radius $R - y$, B_3 with center p and radius x, and, finally, B_4 with center p and radius $x + dx$. Let $\mathcal{V}$ denote the intersection of $(B_1 \setminus B_2)$ and $(B_4 \setminus B_3)$. We prove that the volume of $\mathcal{V}$ is less than $8\,\pi\,x\,y\,dx$.

Since dx is infinitesimally small, the volume of $\mathcal{V}$ is $\mathcal{A}dx$, where $\mathcal{A}$ is the area of the intersection of the sphere ∂B_3 with $B_1 \setminus B_2$.

Let (p, X, Y, Z) be an orthogonal reference frame whose center is p and whose X-axis is oriented along $\overrightarrow{Op}$ (see Figure C.1). Notice that all spheres are centered on that axis. Let C_1 (resp., C_2, C_3) denote the circle that is the boundary of the intersection of B_1 (resp., B_2, B_3) and the plane (p, X, Y) in which Figure C.1 is drawn. The equations of these circles are, in the frame (p, X, Y),

$$\begin{aligned} C_1 : &\quad (X + R - y)^2 + Y^2 = R^2, \\ C_2 : &\quad (X + R - y)^2 + Y^2 = (R - y)^2, \\ C_3 : &\quad X^2 + Y^2 = x^2. \end{aligned}$$

Since C_3 is centered at a point on C_2 and has radius $x \geqslant 6 > 1 \geqslant y$, C_3 intersects or encloses C_1 and C_2. In fact, C_3 intersects or encloses C_1 and C_2 in one of the three following ways.

Case 1. If $6 \leqslant x \leqslant 2R - 2y$, then C_3 intersects both C_1 and C_2 (see Figure C.1(a)).

Case 2. If $2R - 2y < x \leqslant 2R - y$, then C_3 intersects C_1 and encloses C_2 (see Figure C.1(b)).

Case 3. If $2R - y < x$, then C_3 encloses both C_1 and C_2. In that case, $\mathcal{V}$ is empty and the volume is 0.

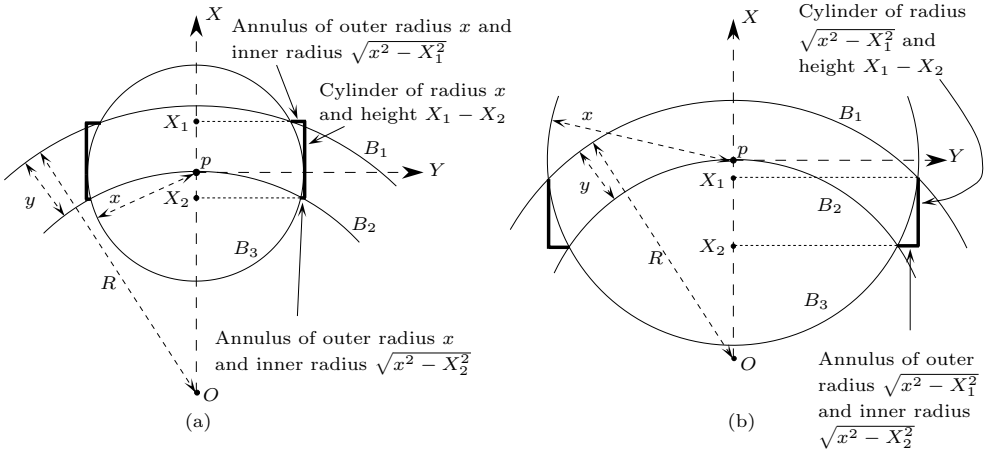


FIG. C.2. *For the proof of Proposition* C.1*, Case* 1.

In the first case, let X_1 (resp., X_2) be the abscissa of the points of intersection of circles C_1 (resp., C_2) and C_3. Note that $X_1 \geqslant X_2$, and their values can be computed directly from the equations of the circles C_1, C_2, and C_3:

$$X_1 = \frac{R^2 - x^2 - (R-y)^2}{2R - 2y}, \qquad X_2 = \frac{-x^2}{2R - 2y}.$$

Using the fact that $y \leqslant x \leqslant 2R - 2y$ we get

$$X_1 - X_2 = \frac{y(2R-y)}{2R-2y} = y\left(1 + \frac{y}{2R-2y}\right) \leqslant 2y,$$
$$-X_1 - X_2 = \frac{2x^2 - y(2R-y)}{2R-2y} \leqslant 2x\frac{x}{2R-2y} \leqslant 2x.$$

We now bound from above the area $\mathcal{A}$ of the surface $\partial B_3 \cap (B_1 \setminus B_2)$ by the area of a larger surface which depends on the sign of X_1. If $X_1 \geqslant 0$, the surface consists of a cylinder of axis, the X-axis, of radius x and height $X_1 - X_2$, and of two annuli in the planes $X = X_1$ and $X = X_2$, of inner radius $\sqrt{x^2 - X_1^2}$ and $\sqrt{x^2 - X_2^2}$, respectively, and outer radius x (see Figure C.2(a)). If $X_1 \leqslant 0$, the surface consists of a cylinder of axis, the X-axis, of radius $\sqrt{x^2 - X_1^2}$ and height $X_1 - X_2$, and of an annulus in the plane $X = X_2$, of inner radius $\sqrt{x^2 - X_2^2}$ and outer radius $\sqrt{x^2 - X_1^2}$ (see Figure C.2(b)). In both cases that surface is larger than $\partial B_3 \cap (B_1 \setminus B_2)$ by convexity.

If $X_1 \geqslant 0$, the area of the cylinder is $2\pi x(X_1 - X_2) \leqslant 4\pi xy$, and the area of the annuli are $\pi x^2 - \pi(x^2 - X_i^2) = \pi X_i^2$, $i = 1, 2$. Since $X_1 \geqslant 0$, $X_1 \leqslant y \leqslant x$, and thus $\pi X_1^2 \leqslant \pi xy$. We also have from the expression of X_1 that $R^2 - x^2 - (R-y)^2 \geqslant 0$, and thus $x^2 \leqslant y(2R - y)$. Thus

$$\pi X_2^2 = \pi \frac{x^2}{2R-2y}\frac{x^2}{2R-2y} \leqslant \pi x \frac{x}{2R-2y} y \frac{2R-y}{2R-2y} = \pi xy \frac{x}{2R-2y}\left(1+\frac{y}{2R-2y}\right).$$

It thus follows from $y \leqslant x \leqslant 2R-2y$ that $\pi X_2^2 \leqslant 2\pi xy$. Hence $\mathcal{A} \leqslant 7\pi xy$.

If $X_1 \leqslant 0$, the area of the cylinder is $2\pi\sqrt{x^2-X_1^2}(X_1-X_2) \leqslant 2\pi x(2y)$, and the area of the annulus is $\pi(x^2-X_1^2)-\pi(x^2-X_2^2) = \pi(X_1-X_2)(-X_2-X_1) \leqslant 4\pi xy$. Thus $\mathcal{A} \leqslant 8\pi xy$.

Consider now the second case $2R-2y < x \leqslant 2R-y$ (see Figure C.1(b)). For a fixed value of y, $\mathcal{A}$ is the area of a spherical cap whose perimeter and curvature decreases as x increases. Thus $\mathcal{A}$ is a decreasing function of x. Since the bound $\mathcal{A} \leqslant 8\pi xy$ is valid for $x = 2R-2y$ and $8\pi xy$ is an increasing function of x, $\mathcal{A} \leqslant 8\pi xy$ for any $x \geqslant 2R-2y$. □

Acknowledgments. The authors would like to thank Helmut Alt, who led us to the lower bound proof of section 4, and Luc Devroye for useful discussions.

REFERENCES

[1] P. K. Agarwal, B. Aronov, and M. Sharir, *Line transversals of balls and smallest enclosing cylinders in three dimensions*, Discrete Comput. Geom., 21 (1999), pp. 373–388.

[2] B. Aronov, H. Brönnimann, A. Chang, and Y.-J. Chiang, *Cost prediction for ray shooting*, in Proceedings of the ACM Symposium on Computational Geometry, Barcelona, Spain, 2002, pp. 293–302.

[3] M. de Berg, H. Everett, and L.J. Guibas, *The union of moving polygonal pseudodiscs – Combinatorial bounds and applications*, Comput. Geom., 11 (1998), pp. 69–81.

[4] F. S. Cho and D. Forsyth, *Interactive ray tracing with the visibility complex*, Comput. Graph., 23 (1999), pp. 703–717.

[5] O. Devillers, B. Mourrain, F. P. Preparata, and P. Trebuchet, *On Circular Cylinders by Four or Five Points in Space*, Rapport de recherche 4195, INRIA, Le Chesnay Cedex, France, 2001.

[6] O. Devillers and P. Ramos, *private communication*, 2001.

[7] F. Durand, *A Multidisciplinary Survey of Visibility*, ACM SIGGRAPH course notes, Visibility, Problems, Techniques, and Applications, 2000.

[8] F. Durand, G. Drettakis, and C. Puech, *The visibility skeleton: A powerful and efficient multi-purpose global visibility tool*, in Proceedings of ACM SIGGRAPH'97, Los Angeles, CA, 1997, pp. 89–100.

[9] F. Durand, G. Drettakis, and C. Puech, *The 3D visibility complex*, ACM Transactions on Graphics, 21 (2002), pp. 176–206.

[10] G. R. Grimmett and D. R. Stirzaker, *Probability and Random Processes*, 2nd ed., Clarendon Press, Oxford, UK, 1992.

[11] N. Holzschuch, F. Sillion, and G. Drettakis, *An efficient progressive refinement strategy for hierarchical radiosity*, in Photorealistic Rendering Techniques, G. Sakas, P. Shirley, and S. Müller, eds., Springer, Berlin, 1995, pp. 353–357.

[12] A. Laurentini, *The visual hull concept for silhouette-based image understanding*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 16 (1994), pp. 150–162.

[13] I.G. Macdonald, J. Pach, and T. Theobald, *Common tangents to four unit balls in* $\mathbb{R}^3$, Discrete Comput. Geom., 26 (2001), pp. 1–17.

[14] *The Maple System*, Waterloo Maple Software, http://www.maplesoft.com/.

[15] A. M. Mathai, *An Introduction to Geometrical Probability: Distributional Aspects with Applications*, Gordon and Breach, Singapore, 1999.

[16] B. Nadler, G. Fibich, S. Lev-Yehudi, and D. Cohen-Or, *A qualitative and quantitative visibility analysis in urban scenes*, Comput. Graph., 23 (1999), pp. 655–666.

[17] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, 3rd ed., McGraw-Hill, New York, 1991.

[18] M. Pellegrini, *Ray shooting on triangles in* 3*-space*, Algorithmica, 9 (1993), pp. 471–494.

[19] S. Petitjean, *The number of views of piecewise-smooth algebraic objects*, in Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 900, Springer, Berlin, 1995, pp. 571–582.

[20] H. Plantinga and C. R. Dyer, *Visibility, occlusion, and the aspect graph*, Internat. J. Comput. Vision, 5 (1990), pp. 137–160.

[21] M. Pocchiola and G. Vegter, *The visibility complex*, Internat. J. Comput. Geom. Appl., 6 (1996), pp. 279–308.
[22] L.A. Santaló, *Integral Geometry and Geometric Probability*, Addison-Wesley, Reading, MA, 1976.
[23] L. Szirmay-Kalos, V. Havran, B. Balazs, and L. Szecsi, *On the efficiency of ray-shooting acceleration schemes*, in Proceedings of the Spring Conference on Computer Graphics, Budmerice, Slovakia, 2002, pp. 89–98.
[24] L. Szirmay-Kalos and G. Márton, *Worst-case versus average case complexity of ray-shooting*, Computing, 61 (1998), pp. 103–131.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1621–1653

PSEUDOTRIANGULATIONS FROM SURFACES AND A NOVEL TYPE OF EDGE FLIP*

OSWIN AICHHOLZER†, FRANZ AURENHAMMER‡, HANNES KRASSER‡, AND PETER BRASS§

Abstract. We prove that planar pseudotriangulations have realizations as polyhedral surfaces in three-space. Two main implications are presented. The spatial embedding leads to a novel flip operation that allows for a drastic reduction of flip distances, especially between (full) triangulations. Moreover, several key results for triangulations, like flipping to optimality, (constrained) Delaunay-hood, and a convex polytope representation, are extended to pseudotriangulations in a natural way.

Key words. pseudotriangulation, flip distance, surface realization, locally convex function, constrained regular complex, polytope representation

AMS subject classifications. 68U05, 68W40, 52C45, 52C25, 52B55, 52B05

DOI. 10.1137/S0097539702411368

1. Introduction.

In geometric data processing, structures that partition the geometric input, as well as connectivity structures for geometric objects, play an important role. A versatile tool in this context is triangular meshes, often called triangulations; see, e.g., the survey articles [14, 26, 11]. A *triangulation* of a finite set S of points in the plane is a maximal planar straight-line graph that uses exactly the points in S as its vertices. Each face in a triangulation is a triangle spanned by S.

In recent years, a relaxation of triangulations, called *pseudotriangulation* (or geodesic triangulation), has received considerable attention. Here, faces bounded by three concave chains, rather than by three line segments, are allowed. Interest in these structures originates from applications to ray shooting [18, 29] and visibility [42, 43]. Meanwhile, their usefulness as an efficient data structure in many other areas has been discovered, including kinetic collision detection [1, 36, 37], rigidity [51, 47, 30], and guarding [44, 49].

Concerning the combinatorial and geometric properties of pseudotriangulations, knowledge is comparatively sparse. Beside the structural rigidity results mentioned above, certain results on the vertex and face degree [35], on the number of possible pseudotriangulations [45, 5], and on edge flipping operations in pseudotriangulations [16, 47] have been obtained recently. An interesting geometric result, in [47] and generalized in [41], addresses polytope representations of pseudotriangulations.

The present paper intends to show that pseudotriangulations possess rich geometric properties. This provides a deeper understanding of their combinatorial properties

*Received by the editors July 15, 2002; accepted for publication (in revised form) May 8, 2003; published electronically October 14, 2003.

http://www.siam.org/journals/sicomp/32-6/41136.html

†Institute for Software Technology, Graz University of Technology, Graz, Austria (oaich@ist.tugraz.at). This work was done while this author was with the Institute for Theoretical Computer Science, Graz University of Technology, Graz, Austria. This author's research was supported by the Austrian Programme for Advanced Research and Technology (APART).

‡Institute for Theoretical Computer Science, Graz University of Technology, Graz, Austria (auren@igi.tugraz.at, hkrasser@igi.tugraz.at). The research of these authors was supported by the Austrian Fonds zur Förderung der Wissenschaftlichen Forschung (FWF).

§Department of Computer Science, City College of New York, New York (peter@cs.ccny.cuny.edu). This work was done while this author was with the Institut für Informatik, Freie Universität Berlin, Berlin, Germany. This author's research was supported by DFG under grant Br 1465/5-2.

and of the interrelation of pseudotriangulations of different levels of edge rank. One of our main results asserts that pseudotriangulations have realizations as polyhedral surfaces in three-space. This insight leads to a novel edge flip operation, which has a three-dimensional interpretation in the surface, as well as a consistent geodesics interpretation in the plane. Inclusion of the new flip type allows us to transform arbitrary pseudotriangulations into each other without changing the underlying set of vertices. A tool for rapidly adapting pseudotriangulations (in particular, triangulations) and their realizing surfaces becomes possible, using a near-linear number of constant-size combinatorial changes. Pseudotriangulations that are realizable as locally convex surfaces project to unique constrained regular complexes, a concept which has not been rigorously defined before. We further solve the optimization problem of finding maximal locally convex surfaces over polygonal domains, by bounding the length of (improving) flip sequences that reach the global optimum of local convexity. Finally, we provide a polytope representation of constrained regular pseudotriangulations.

Subsection 1.1 gives a more detailed overview of our results. We expect various applications in areas where (pseudo)triangulations and polyhedral surfaces play a role, such as surface modeling and terrain fitting. Related concepts, which may be affected by our results, are power diagrams [8] (a generalization of Voronoi diagrams, dual to regular triangulations), weighted alpha shapes [24] and dual complexes [23] (certain subcomplexes of regular triangulations), reflex-free hulls [2] (polyhedral surfaces avoiding cavities), and roofs [3] (surface representations of straight skeletons). The focus of the present paper, though, is on providing new and fundamental properties of pseudotriangulations, rather than on a particular application.

1.1. Overview of results. We start by revisiting edge flipping operations in pseudotriangulations, in section 2. A new type, the edge-removing flip, is introduced, and its relationship to existing flip types is discussed. Section 3 shows that, when using the edge-removing flip, the flip distance between any two pseudotriangulations (and especially, triangulations) of a set of n points is reduced to $O(n \log n)$. This is a substantial improvement over the situation without the new flip type, where an $\Omega(n^2)$ lower bound holds for triangulations [31, 34]. We further derive an $O(n \log^2 n)$ bound for minimum pseudotriangulations, *without* using edge-removing flips. This improves the previous bounds of $O(n^2)$ in [16, 47] and shows that the diameter of the high-dimensional polytope in [47] is $O(n \log^2 n)$. For special settings, even some linear flip distance bounds are achieved. The results above partially rely on new partitioning results for pseudotriangulations in section 4, which may be of separate interest.

Section 5 describes realizations of pseudotriangulations as polyhedral surfaces. We introduce the concept of complete vertices, to distinguish among the vertices those whose heights in the surface can be predetermined. We characterize the class of pseudotriangulations which are projective, in the sense that their internal edges exactly correspond to the set of nonlinearity of a polyhedral surface. This generalizes the situation for triangulations, which are trivially projective. Section 6 gives a surface interpretation of all the admissible flip types from section 2. Surface flips, while being constant-size combinatorial changes, are geometrically nonlocal; a single flip may change the surface heights for $\Theta(n)$ vertices. Special cases are tetrahedral surface flips, which are well known from flipping in convex hulls; see, e.g., [38, 25].

Section 7 puts the above results to use, by solving the optimization problem of finding maximal locally convex functions on polygonal domains (with n vertices, some possibly internal). We prove that every triangular surface can be made locally convex, by performing surface flips which are convexifying or planarizing. The resulting pro-

jection is a pseudotriangulation. If the underlying domain is convex, or if this domain contains no internal vertices, then $O(n^2)$ flips suffice. The former result strengthens a result in [25], which shows that incrementally inserting the vertices, and flipping away all reflex edges in the special (triangular) surface that arises from each insertion, is capable of generating the optimum. For convex domains, the optimum is a lower convex hull, and a regular triangulation in the projection. It is well known [25] that the optimum cannot be reached *within* the class of triangulations, in general. This explains the need for edge-removing (i.e., planarizing) flips and substantiates the usefulness of pseudotriangulations. Our results can also be viewed as a generalization of a well-known fact, namely, that every triangulation of a planar point set can be transformed into the Delaunay triangulation, using so-called Delaunay flips; see, e.g., [26].

Extending the scenario, by fixing a set of edges which are not to be flipped, we arrive at the concept of constrained regular complexes, in subsection 8.2. These are projections of unique maximal surfaces that are locally convex with respect to a constraining planar straight-line graph. If this graph is connected, then pseudotriangulations are obtained. In contrast, constrained regular *triangulations* do not always exist [13, 50] (or are ambiguous structures, depending on the definition), apart from special cases like the constrained Delaunay triangulation [40, 19]. The algorithmic results from section 7 carry over to the constrained setting. In section 9, we show the existence of a convex polytope in n dimensions, whose vertices correspond to all the regular pseudotriangulations constrained by a given graph. The edges of this polytope represent flips of the admissible types. This implies that the set of constrained regular pseudotriangulations is connected under admissible flip operations, and generalizes the polytope constructions in [39, 28, 15] (the so-called associahedron, or secondary polytope) for regular triangulations. As a consequence, locally convex surfaces can be deformed into each other in a controlled way using surface flips.

1.2. Basic properties of pseudotriangulations. We first review some basic notions concerning and properties of pseudotriangulations. For more details, we refer to, e.g., [35, 16, 47]. A *(simple) polygon* is a subset of the plane, homeomorphic to a disk and with piecewise-linear boundary. We denote with $vert(P)$ the set of vertices of a polygon P. A *corner* of P is a vertex with internal angle less than π. The other vertices of P are called *noncorners*. A *side chain* of P is the chain of edges between two consecutive corners of P. The *geodesic* between two points $x, y \in P$ is the shortest curve that connects x and y and lies inside P. A *pseudotriangle* ∇ is a polygon with exactly three corners. Note that the corners of ∇ are the vertices of the convex hull $conv(\nabla)$ of ∇. Moreover, for each pair of corners of ∇, the geodesic between them entirely lies on ∇'s boundary and defines a side chain of ∇.

Let S be a finite set of points in the plane. We will assume throughout this paper that S is in general position (i.e., no three points in S are collinear). A *pseudotriangulation* of S is a partition of $conv(S)$ into pseudotriangles whose vertex set is exactly S. A pseudotriangulation is a face-to-face two-dimensional cell complex; see Figure 1.1 for an illustration. Two faces (pseudotriangles) may be adjacent at one or two edges. In case of such *double-adjacencies*, the union of the two pseudotriangles is a pseudotriangle itself.

Let $\mathcal{PT}$ be some pseudotriangulation of S. A vertex of $\mathcal{PT}$ is called *pointed* if its incident edges lie in an angle smaller than π. Note that all vertices of $conv(S)$ are pointed. The more pointed vertices there are in $\mathcal{PT}$, the less edges and faces it has. More precisely, $\mathcal{PT}$ contains exactly $3n - p - 3$ edges and $2n - p - 2$ pseudotriangles

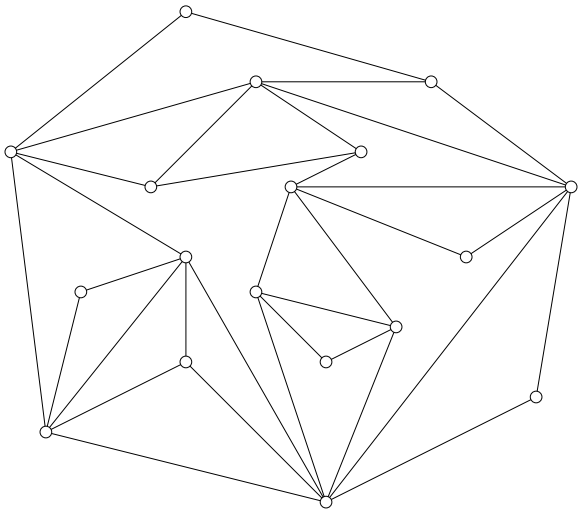


FIG. 1.1. *A pseudotriangulation for* 18 *points.*

if $|S| = n$ and there are $p \leq n$ pointed vertices in $\mathcal{PT}$.

We define the *edge rank* of $\mathcal{PT}$ as $n - p$. The minimum edge rank is zero, where $\mathcal{PT}$ is commonly called a *minimum* (or a *pointed*) pseudotriangulation. $\mathcal{PT}$ then is a maximal planar straight-line graph on S where all vertices are pointed; see, e.g., [51]. It contains exactly $2n - 3$ edges and $n - 2$ pseudotriangles. The edge rank expresses the excess in edges, compared to a minimum pseudotriangulation. Its value is at most $n - |vert(conv(S))|$, which is attained if and only if $\mathcal{PT}$ is a triangulation. The reader may verify that the pseudotriangulation in Figure 1.1 has edge rank 1.

2. Flips in pseudotriangulations revisited. So-called *flips* are operations of constant combinatorial complexity which are commonly used to modify triangulations [4, 25, 26, 31, 34, 38] and, as of late, pseudotriangulations [43, 36, 16, 47]. The purpose of the present section is to introduce a novel flip operation and to show its natural relationship to existing flip types.

2.1. Classical flips. The standard edge flip, also called the *Lawson flip* [38], takes two triangles Δ_1 and Δ_2 whose union is a convex quadrilateral and exchanges the diagonals e and e' of this quadrilateral. To generalize to pseudotriangulations, a different view of this edge flip is advantageous: Take the vertex of Δ_1 and Δ_2, respectively, that lies opposite to e, and replace e by the geodesic between these two vertices. The geodesic is just a line segment e' in this case.

The geodesics interpretation above has been used in [42, 51] to define flips in minimum pseudotriangulations. Let ∇_1 and ∇_2 be two adjacent pseudotriangles, and let e be an edge they have in common. A flip replaces e by the part of the geodesic interior to $\nabla_1 \cup \nabla_2$ that connects the two corners of ∇_1 and ∇_2 opposite to e. In a minimum pseudotriangulation each vertex is pointed, and thus the geodesic indeed contributes a line segment e' which is no edge of ∇_1 or ∇_2.

See Figure 2.1(a) and (b), where the edge e to be flipped is shown in bold and the obtained geodesic is drawn dashed. Note that the flipping partners e and e' may cross or not. In either case, the flip creates two valid pseudotriangles. We refer to such flips as *exchanging* flips. In a minimum pseudotriangulation, each internal

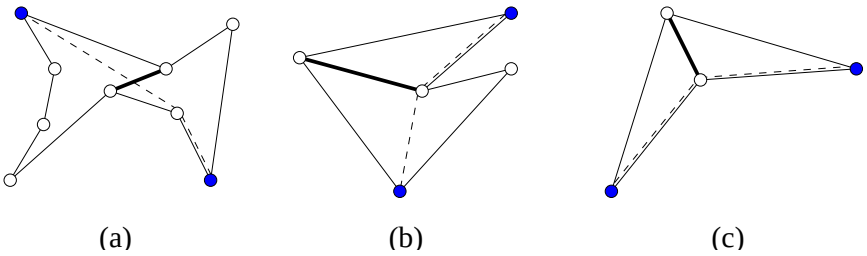


FIG. 2.1. *Exchanging flips and nonflippable edge.*

edge can be flipped to a unique different edge. In a pseudotriangulation of nonzero edge rank, however, edges incident to nonpointed vertices may be *nonflippable* in this sense. In particular, in a full triangulation, an internal edge is nonflippable if and only if its two incident triangles form a nonconvex quadrilateral; see Figure 2.1(c). Nonflippable edges have been the source for the theoretically poor behavior of certain flipping algorithms, concerning the flip distance [31, 34] as well as the nonexistence of flip sequences [25].

2.2. A novel flip type. We wish to generalize the edge flip so as to cover the situation in Figure 2.1(c) as well. In fact, when being consistent with the geodesics rule above, flipping a nonflippable edge $e = \nabla_1 \cap \nabla_2$ means removing e because its substitute does not exist. A pseudotriangle $\nabla_1 \cup \nabla_2$ is obtained. We include this *edge-removing* flip (and its inverse, the *edge-inserting* flip) into the repertoire of admissible flips. By definition, an edge-removing flip is applicable only if a valid pseudotriangle is created. That is, a single nonpointed vertex of the pseudotriangulation is made pointed by the flip.

This simple modification makes each internal edge in every pseudotriangulation (and, in particular, in every triangulation) flippable. Note that each edge-removing flip decreases the edge rank by one, whereas each edge-inserting flip increases it by one. This allows for "surfing" between pseudotriangulations of different edge ranks. Various interesting consequences will be discussed, including a reduction of flip distances, in section 3, and a three-dimensional interpretation of the new flip type, in section 6.

2.3. Relation to other types. The new flip type can be used to simulate certain other flip types. For instance, the exchanging flip in Figure 2.1(b) can be simulated by an edge-inserting flip followed by an edge-removing flip. Of particular relevance is a type of flip that arises in the context of Delaunay and regular triangulations; see [46, 25]. This flip inserts a new vertex v into the interior of a triangle Δ and connects v by edges to the three vertices of Δ. Vertex insertion is meaningful for pseudotriangulations as well [51, 5]. Connect v by geodesics to (at least two) corners of the pseudotriangle ∇ in which v lies. Each geodesic contributes one edge incident to v, and ∇ is partitioned into (two or three) pseudotriangles.

Let us simulate the inverse operation, the removal of a degree-3 vertex v, using the new flip type; see Figure 2.2. Apply an edge-removing flip to one of v's edges first, which leaves a partition of ∇ into two pseudotriangles in double-adjacency. Then, carry out two edge-removing flips simultaneously. This deletes v and leaves ∇ empty, because no edges are created by the geodesics rule. We consider the simultaneous operation as a single flip—the *vertex-removing* flip. By definition, vertex-removing

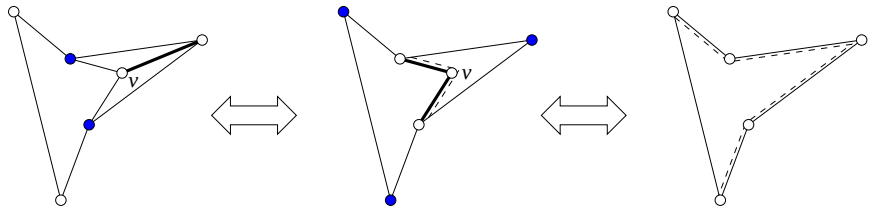


FIG. 2.2. *Edge-removing and vertex-removing flips.*

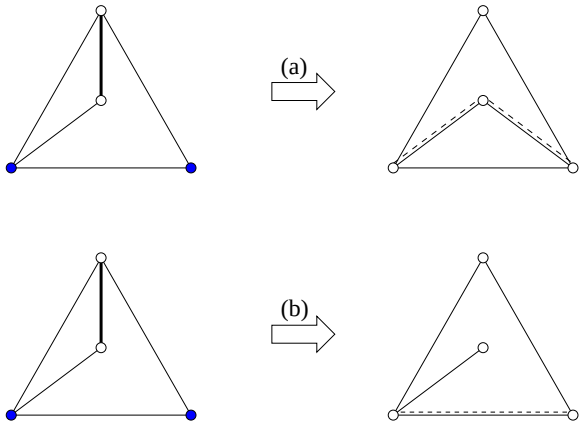


FIG. 2.3. *Ambiguous geodesics interpretation.*

flips can be applied to vertices of degree 2 only.

In summary, the repertoire of flip operations we are going to use contains the (classical) exchanging flips, the edge-removing and the vertex-removing flip, and their inverses, the inserting flips.

Remarks. Edge-removing flips arise implicitly in a greedy flip algorithm for pseudotriangulations of *convex objects*, in [43]. Certain flips that exchange bitangents of such objects cause an edge removal (or insertion) in the corresponding pseudotriangulation for the object centers.[1]

Instead of the vertex-removing flip, a different version—namely, the exchanging flip in Figure 2.3(a)—has been commonly used. It also leads to a valid pseudotriangulation (which now does contain the vertex v). However, care has to be taken not to misinterpret this version as in Figure 2.3(b), where the geodesic still lies inside the union of the two pseudotriangles involved. Also, this version conflicts with a three-dimensional interpretation of flips in surfaces, in section 6. When change in edge rank is conceded, we may circumvent the flip in Figure 2.3(a) by performing two consecutive flips of the new type, namely, an edge-inserting flip followed by an edge-removing flip.

The vertex-inserting flip together with the exchanging flips are the constituting operations of the so-called Henneberg construction, which is capable of generating every minimum pseudotriangulation of a point set S; see, e.g., [51]. With our reper-

[1] We recently learned that edge-removing flips have been introduced independently (and at about the same time) in [41], to generalize the polytope representation in [47] to pseudotriangulations of arbitrary edge rank. In fact, Theorem 3.4 in section 3 implies a bound of $O(n \log n)$ on the diameter of their polytope.

toire, arbitrary pseudotriangulations of S can be derived from each other without ever changing S. We will elaborate on this issue in detail in section 3.

3. Reducing the flip distance. Let S be a set of n points in the plane. It is well known that $\Theta(n^2)$ Lawson flips may be necessary, and are also sufficient, to transform two given triangulations of S into each other; see, e.g., [31, 34]. The upper bound also applies to exchanging flips in minimum pseudotriangulations (see [16, 47]), but no nontrivial lower bounds are known in this case. This section presents a series of improved upper bounds on flip distances, partially even without using the new flip type.

3.1. Simple polygons. We start with proving that flip distances become linear between pseudotriangulations of (simple) polygons when edge-removing flips and their inverses are allowed. Consider a polygon P in the plane. The *shortest-path tree* of P with root $v \in vert(P)$ is the union of all geodesics in P from $vert(P)$ to v. Let $\pi_v(P)$ denote this structure. It is well known [32] that $\pi_v(P)$ is a tree that partitions P into pseudotriangles in a unique way.

LEMMA 3.1. *Let P be a polygon with n vertices, and let $v \in vert(P)$. The shortest-path tree $\pi_v(P)$ can be constructed by triangulating P arbitrarily and applying at most $n-3$ exchanging or edge-removing flips.*

Proof. Fix some triangulation $\mathcal{T}$ of P. We prove the assertion by induction on the number of triangles of $\mathcal{T}$. As an induction base, let Q be the union of all triangles of $\mathcal{T}$ incident to the vertex v. Clearly, the restriction of $\mathcal{T}$ to Q just gives $\pi_v(Q)$. We show that this invariant can be maintained by flipping, when an adjacent triangle Δ of $\mathcal{T}$ is added to Q.

Let u be the vertex of Δ that does not belong to Q. Consider the unique edge $e = Q \cap \Delta$ (which is a diagonal of P). If e belongs to $\pi_v(Q \cup \Delta)$, then an edge of Δ connects u to $\pi_v(Q)$, and $\pi_v(Q \cup \Delta)$ is already complete. No flip is performed. Else let ∇ denote the unique pseudotriangle in $\pi_v(Q)$ that is adjacent to Δ at e. There are two cases. If $\nabla \cup \Delta$ is a pseudotriangle, then, again, u is connected to $\pi_v(Q)$ by Δ. Perform a flip that removes e, which restores $\pi_v(Q \cup \Delta)$. Otherwise, let w be the corner of ∇ opposite to e. Apply an exchanging flip to e. The new edge e' lies on the geodesic between u and w. Thus e' connects u to $\pi_v(Q)$, which constructs $\pi_v(Q \cup \Delta)$ in this case.

The total number of flips is at most $n-3$, because each flip can be charged to the triangle of $\mathcal{T}$ that is added. □

COROLLARY 3.2. *Any two triangulations of a polygon P with n vertices can be flipped into each other by at most $2n-6$ exchanging, edge-removing, or edge-inserting flips.*

Proof. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two triangulations of P. Choose some vertex $v \in vert(P)$ and flip $\mathcal{T}_1$ to $\pi_v(P)$. Then flip $\pi_v(P)$ to $\mathcal{T}_2$ by reversing the sequence of flips that transforms $\mathcal{T}_2$ to $\pi_v(P)$. This is possible and takes at most $2n-6$ flips of the types above, by Lemma 3.1. □

Corollary 3.2 implies a flip distance of $O(n)$ between any two *pseudotriangulations* $\mathcal{PT}_1$ and $\mathcal{PT}_2$ of a polygon P, because $\mathcal{PT}_1$ and $\mathcal{PT}_2$ can be completed to form a triangulation of P with $O(n)$ edge-inserting flips.

3.2. Planar point sets. We continue with pseudotriangulations of planar point sets. In fact, we choose a slightly more general scenario, namely, a point set enclosed by an arbitrary simple polygon (a so-called augmented polygon). This setting will turn out to be more appropriate for our developments, as it arises naturally from

constraining the pseudotriangulated domain. We will show how to flip any given pseudotriangulation into a canonical one by splitting the underlying augmented polygon in a balanced way until empty polygons are obtained and Corollary 3.2 applies.

Let P be a polygon, and consider a finite point set $S \subset P$ with $vert(P) \subseteq S$. We call the pair (P, S) an *augmented polygon*. A pseudotriangulation $\mathcal{PT}$ of (P, S) is a partition of P into pseudotriangles whose vertex set is exactly S. It contains exactly $3n - m + k - p - 3$ edges and $2n - m + k - p - 2$ pseudotriangles if $|S| = n$, P is an m-gon with k corners, and p counts the pointed vertices of $\mathcal{PT}$. The maximum edge rank of $\mathcal{PT}$ is $n - k$, in which case $\mathcal{PT}$ is a triangulation. In the special case $P = conv(S)$, we have $m = k$ and deal with pseudotriangulations of the point set S. Below is another corollary of Lemma 3.1.

COROLLARY 3.3. *Let $\mathcal{T}$ be a (full) triangulation of an augmented polygon (P, S). Let e be some line segment spanned by S, which lies inside P and crosses $\mathcal{T}$ at $j \geq 1$ edges. Then $\mathcal{T}$ can be modified to a triangulation that contains e by applying $O(j)$ exchanging, edge-removing, or edge-inserting flips.*

Proof. Let Q be the union of the triangles of $\mathcal{T}$ that are crossed by e. Note that Q may contain points of S in its interior or even may contain holes, namely if Q contains internal edges which do not cross e. In this case, we cut Q along these edges and move Q apart infinitesimally at the cuts, to obtain a simple polygon empty of points in S. This is possible because a general position is assumed for S. Now choose any triangulation $\mathcal{T}_e$ of Q which includes the edge e to be integrated. By Corollary 3.2, the part of $\mathcal{T}$ inside Q can be flipped to $\mathcal{T}_e$ by $O(j)$ flips. □

We are now prepared to prove the following general assertion on flip distances.

THEOREM 3.4. *Any two pseudotriangulations of a given planar point set S (or more generally, of a given augmented polygon (P, S)) can be transformed into each other by applying $O(n \log n)$ flips of the types exchanging, edge-removing, and edge-inserting, for $n = |S|$. No vertex-removing flips are used.*

Proof. The two pseudotriangulations of the augmented polygon (P, S) in question can be completed to form a triangulation by applying $O(n)$ edge-inserting flips. We show how to transform two arbitrary triangulations $\mathcal{T}_1$ and $\mathcal{T}_2$ of (P, S) into the same, using $O(n \log n)$ flips. Let P be an m-gon. If $m = n$, then $O(n)$ flips suffice, by Corollary 3.2. Else we partition P into subpolygons, each containing at most $\frac{2}{3}(n - m)$ points of $S \setminus vert(P)$. A constant number of line segments spanned by S suffice for this purpose, by Theorem 4.4(1) below. Incorporate these segments into $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively, in $O(n)$ flips, which is possible by Corollary 3.3. Treat the obtained $O(1)$ augmented polygons recursively. This yields a polygonal partition of P whose vertex set is exactly S, and two triangulations thereof, in $O(n \log n)$ flips. By Corollary 3.2, another $O(n)$ flips let these two triangulations coincide. □

Remarks. Theorem 3.4 demonstrates that flip distances are substantially reduced when using the new flip type. "Shortcuts" via pseudotriangulations with varying edge rank become possible. The interested reader may check that the constant involved in the $O(n \log n)$ term is small (less than 6). We conjecture that Theorem 3.4 can be improved to $O(n)$ flips, because the $\Omega(n^2)$ worst-case examples for Lawson flips in triangulations are based on (nonconvex) polygons without internal points [34], an instance covered by Corollary 3.2 in $O(n)$ flips.

All flips used in Theorem 3.4 are constant-size combinatorial operations, which can be carried out in $O(\log m)$ time each if the size of the two pseudotriangles involved is at most m; see, e.g., [27]. This implies that any two (pseudo)triangulations of a given set of n points can be adapted by local operations in $O(n \log^2 n)$ time—a result

we expect to have various applications.

3.3. Two linear bounds. For the sake of completeness, we include here two more implications that come from using our extended repertoire of flips.

Recall that vertex-removing and vertex-inserting flips have been excluded in Theorem 3.4. Dropping this restriction makes things easy, because every pseudotriangulation contains some vertex of constant degree, which can be removed with $O(1)$ flips. A flip distance of $O(n)$ is obvious in this setting. However, removing a vertex not only changes the vertex set S, but also changes the underlying domain (the polygon P) if removal was for a boundary vertex. In contrast, in the setting in Theorem 3.4, both S and P remain unchanged. The situation where S but not P is allowed to change is of some interest, because no internal vertex of constant degree might exist in a pseudotriangulation.

LEMMA 3.5. *Let S be a planar n-point set. Any two pseudotriangulations of S can be flipped into each other in $O(n)$ flips, without changing the underlying domain $conv(S)$, if the entire repertoire of flips from section 2 is used.*

It suffices to prove Lemma 3.5 for full triangulations. Let $\mathcal{T}^*$ be the *star triangulation* of $conv(S)$ with respect to a fixed boundary vertex v. In $\mathcal{T}^*$, each vertex of $conv(S)$ different from v is connected to v by an edge. We show that any triangulation $\mathcal{T}$ of S can be flipped into $\mathcal{T}^*$ with $O(n)$ flips. To this end, consider a particular order for the triangles of $\mathcal{T}$.

LEMMA 3.6. *Let v be some boundary vertex of $\mathcal{T}$. There exists an order $\Delta_1, \ldots, \Delta_t$ for $\mathcal{T}$'s triangles such that $\bigcup_{i \le j} \Delta_j$ is star-shaped as seen from v, for $j = 1, \ldots, t$.*

Proof. Set $Q_j = \bigcup_{i \le j} \Delta_j$. To apply induction, assume that Q_j is star-shaped as seen from v, for some j; this assumption is obviously true for $Q_t = conv(S)$. Let Δ be a triangle of T in Q_j that is not incident to v. The part of Δ on the boundary of Q_j (if existent) defines a visibility angle $\alpha(\Delta)$ at v. Because Q_j is star-shaped and v is a boundary vertex of Q_j, there exists some triangle Δ which lies within $\alpha(e)$. Thus $Q_j \setminus \Delta = Q_{j-1}$ is star-shaped too. □

We now visit $\Delta_1, \ldots, \Delta_t$ in the order given by Lemma 3.6, and maintain the star triangulation T_j^* of Q_j with respect to v by flipping. If the current triangle Δ_j is incident to v, then no flip is performed. Otherwise, Δ_j shares either one or two edges with Q_{j-1}. Let e be such an edge. In the former case, the exchanging flip for e restores T_j^*. To obtain T_j^* in the latter case, the flip removing e is applied first, followed by a flip that removes the unique vertex w internal to Q_j. (Note that the degree of w has been lowered to 2 by the removal of e.) This yields $T_t^* = \mathcal{T}^*$ after less than $2t = O(n)$ flips and proves Lemma 3.5.

The next linear bound concerns minimum pseudotriangulations. It is well known that not every pseudotriangulation can be made minimum by removing edges. It can only be made *minimal* in edge rank, and is termed accordingly in [1, 35, 48]. In particular, a minimal pseudotriangulation may be a full triangulation, even when its vertices are not in convex position [48]. We can show the following result.

LEMMA 3.7. *Let $\mathcal{PT}$ be any pseudotriangulation of a planar n-point set S. Then $\mathcal{PT}$ can be transformed into a minimum pseudotriangulation of S with $O(n)$ exchanging, edge-removing, or edge-inserting flips. No vertex-removing flips are used.*

Proof. Consider some spanning tree B of $\mathcal{PT}$. We construct a connected graph Z with vertex set S, which does not cross B and where each vertex is pointed. Let $v_1, \ldots, v_n$ be the vertices of S in some depth-first order for B such that, for each v_i, the edges of B incident to v_i are visited in clockwise order. To obtain Z, connect v_i

and v_{i+1} for $i = 1, \ldots, n$ and $v_{n+1} = v_1$, as follows. If $v_i v_{i+1}$ is an edge of B, then use this edge. Otherwise, use the geodesic that comes from "stretching" the unique path in B between v_i and v_{i+1} to the shortest path in the plane that does not cross B. By construction, each vertex v_i either has degree 2 in Z or some geodesics run via v_i, in which case v_i is pointed in Z as well.

In general, $\mathcal{PT}$ will not contain Z as a subgraph. We flip $\mathcal{PT}$ into a pseudotriangulation $\mathcal{PT}'$ that contains Z (and B) as follows. Imagine cutting $conv(S)$ along B and moving it apart at the cuts infinitesimally. This yields a partition of $conv(S)$ into simple polygons. Within each such polygon, $\mathcal{PT}$ can be flipped to contain the relevant part of Z, in a total of $O(n)$ flips for all polygons; see subsection 3.1. Finally, we flip $\mathcal{PT}'$ to a minimum pseudotriangulation that contains Z, by cutting $conv(S)$ along Z and adapting the pseudotriangulations within these polygons, using $O(n)$ additional flips. □

Remarks. The order in Lemma 3.6 is called a shelling order (or a translation order) for $\mathcal{T}$ with respect to v. It is well known that such an order does not exist for every triangulation if v is an *internal* vertex. Exceptions are Delaunay triangulations [21] and regular triangulations [22].

Lemma 3.7 does not imply a linear flip distance between any two pseudotriangulations of S, because which minimum pseudotriangulations can be reached depends on $\mathcal{PT}$. The pointed graph Z arises in a classical factor-2 approximation for the travelling salesman tour for S. We raise the question of whether every triangulation of a point set S contains a pointed spanning tree.

3.4. Minimum pseudotriangulations. Our next aim is to provide a stronger version of Theorem 3.4, namely for minimum pseudotriangulations and when using the exchanging flip type exclusively. That is, we restrict ourselves to staying within the class of minimum pseudotriangulations. We apply arguments similar to those of subsections 3.1 and 3.2.

Let P be a polygon with k corners. The *minimum shortest-path tree* $\mu_c(P)$, for a fixed corner c of P, is the union of all geodesics inside P that lead from c to the *corners* of P. Observe that $\mu_c(P)$ defines a minimum pseudotriangulation for P, which is a subset of the shortest-path tree $\pi_c(P)$ defined in subsection 3.1. The proof of Lemma 3.1 now can be adapted to show that every minimum pseudotriangulation of P can be transformed into $\mu_c(P)$ by at most $k-3$ exchanging flips. The new flip type is not used here, because each internal edge is flippable in the classical sense. We obtain the following.

LEMMA 3.8. *Let P be a polygon with k corners. Any two minimum pseudotriangulations of P are transformable into each other by applying at most $2k-6$ exchanging flips.*

The next lemma is a variant of Corollary 3.3 for minimum pseudotriangulations.

LEMMA 3.9. *Let $\mathcal{MPT}$ be a minimum pseudotriangulation of an augmented polygon (P, S), and let G be some pointed planar straight-line graph on S and in P. Then $\mathcal{MPT}$ can be modified to contain G by applying $O(nj)$ exchanging flips if S has n vertices and $G \setminus P$ has j edges.*

Proof. Let uv be some edge of $G \setminus P$. We prove that uv can be incorporated into $\mathcal{MPT}$ using $O(n)$ exchanging flips, and without altering previously incorporated edges of G.

If vertex u is not pointed in $\mathcal{MPT} \cup \{uv\}$, we flip edges of $\mathcal{MPT}$ incident to u to restore the pointedness of u, as follows. Let $m(u)$ denote the angle formed by u's edges in the current minimum pseudotriangulation, and let $g(u)$ be the angle

formed by those edges among them which also belong to G (if there are any). We have $g(u) < m(u) < \pi$. Rotate $m(u)$, by flipping the unique edge e that bounds $m(u)$ but not $g(u)$ and such that e and $g(u)$ lie on different sides of uv, until $uv \in m(u)$. The condition is necessarily met as soon as e does not exist. This process does not alter any edge of G in $\mathcal{MPT}$, takes $O(n)$ exchanging flips, and transforms $\mathcal{MPT}$ into a minimum pseudotriangulation $\mathcal{MPT}'$. Note that the vertex v may be nonpointed in $\mathcal{MPT}' \cup \{uv\}$.

We next flip the edges of $\mathcal{MPT}'$ which cross uv (if there are any), in the order from u to v. For the current edge e to be flipped, the following invariants hold. Edge e bounds a face ∇ incident to u. Denote with c the corner of ∇ opposite to e. ($c = u$ is possible.) The edge f obtained by flipping e lies on some geodesic to c. Thus f leaves u pointed. Moreover, as the line segment uv also lies on a geodesic to c, edge f does not cross uv. After $O(n)$ flips, a minimum pseudotriangulation $\mathcal{MPT}''$ results, where no edge crosses uv and such that u is still pointed in $\mathcal{MPT}'' \cup \{uv\}$.

Now either $uv \in \mathcal{MPT}''$ (and we are done) or v is not pointed in $\mathcal{MPT}'' \cup \{uv\}$. In the latter case, there is a unique face ∇'' of $\mathcal{MPT}''$ which contains uv in its interior, and v is a noncorner of ∇''. Consider the side chain of ∇'' that contains v, and let c'' be the corner of ∇'' opposite to this side chain. Flip an edge of ∇'' which is incident to v and which does not lie in the pointed angle of v in G. The obtained edge f'' lies on a geodesic to the corner c''. Therefore, f'' neither crosses uv nor destroys the pointedness of u. If $f'' = uv$, the construction is complete. Otherwise, the degree of v has been decremented in the flip, and we repeat the step above for the face that now contains uv in its interior, until uv becomes part of the current pseudotriangulation. This takes another $O(n)$ flips. □

Lemma 3.9 directly implies an $O(n^2)$ bound on the (exchanging) flip distance in minimum pseudotriangulations: Take for G the minimum pseudotriangulation into which $\mathcal{MPT}$ is to be transformed. The proof of the theorem below shows that Lemma 3.9 can be used in a more sophisticated way.

THEOREM 3.10. *Let S be a set of n points in the plane, and let $\mathcal{MPT}_1$ and $\mathcal{MPT}_2$ be two minimum pseudotriangulations of S. Then $\mathcal{MPT}_1$ can be transformed into $\mathcal{MPT}_2$ by applying $O(n \log^2 n)$ exchanging flips. No other flip types are used. The same result holds for augmented polygons (P, S).*

Proof. Consider an augmented polygon (P, S). We recursively split (P, S) in a balanced way, by applying Theorem 4.4(2) from section 4. This constructs a polygonal partition Π of P, whose vertex set is S and where all vertices are pointed. Π is obtained by introducing $O(\log n)$ line segments spanned by S in each recursive step, and the number of recursive steps is $O(\log n)$.

By Lemma 3.9, $O(n \log^2 n)$ exchanging flips are sufficient to make $\mathcal{MPT}_1$ and $\mathcal{MPT}_2$ contain all the edges of Π. Finally, Lemma 3.8 allows for adapting pseudotriangles within the polygons of Π in $O(n)$ such flips. □

Remarks. Theorem 3.10 improves the recent bound of $O(n^2)$ in [16] for minimum pseudotriangulations of point sets. Again, we conjecture that the truth is $O(n)$ flips.[2]

In [47], the polytope $\mathcal{M}(S)$ of minimum pseudotriangulations of a point set S was introduced. $\mathcal{M}(S)$ is a high-dimensional convex polytope. Its vertices correspond to all the minimum pseudotriangulations of S, and its edges represent all possible exchanging flips. By Theorem 3.10, the diameter of $\mathcal{M}(S)$ is bounded by $O(n \log^2 n)$.

[2] By a very recent result in [12], a flip distance of $O(n \log n)$ for minimum pseudotriangulations of point sets is obtainable, using a different divide-and-conquer approach. However, this approach does not carry over to the more general case of augmented polygons.

As a somewhat counterintuitive fact, the transformation between two given minimum pseudotriangulations may be speeded up by using edge-inserting and edge-removing flips intermediately. There exist examples where $n-3$ exchanging flips are necessary, a third of which are saved when performing a single edge-inserting and edge-removing flip, respectively, in the beginning and at the end; see [33]. This indicates that flexibility of pseudotriangulations not only comes from low edge rank, but also stems from the ability to change this parameter, using the new flip type.

A *constrained* pseudotriangulation of a point set S is one that is required to contain a given planar straight-line graph G on S. It is well known that, for any choice of G, the set of constrained *triangulations* of S is connected by Lawson flips: Every triangulation constrained by G can be flipped into the constrained Delaunay triangulation for G and S; see [40]. The following related result holds, by arguments in the proof of Lemma 3.9.

COROLLARY 3.11. *Any two minimum pseudotriangulations of a point set S constrained by a given (pointed) planar straight-line graph G on S can be flipped into each other using $O(n^2)$ exchanging flips.*

In case G partitions $conv(S)$ into augmented polygons (as in Figure 8.1 below), the $O(n \log^2 n)$ exchanging flip distance bound from Theorem 3.10 applies. Moreover, for such graphs G, the flip distance between constrained pseudotriangulations of arbitrary edge rank is $O(n \log n)$ when edge-inserting flips and their reverses are admitted, by Theorem 3.4. The concept of constrained pseudotriangulations plays an important role in the context of regularity; see sections 8 and 9.

4. Partitioning results. This section presents some partitioning results concerning pseudotriangulations that were referred to in section 3. Subsections 4.1 and 4.3 give two technical lemmas. The assertions in subsections 4.2 and 4.4 might be of separate interest.

4.1. Pseudoconvex polygons. Let P be a simple polygon. Consider a pseudotriangle $\nabla \subset P$ with vertices from $vert(P)$. ∇ is called *nice* if its three corners are corners of P. We define the *cut* of ∇ as the number of diagonals of P on ∇'s boundary.

A polygon P is *pseudoconvex* if every geodesic inside P is a convex chain. A *corner tangent* is an inner tangent of P incident to at least one corner of P. A pseudoconvex polygon P is *strongly pseudoconvex* if no corner tangent exists for P.

LEMMA 4.1. *Let P be a strongly pseudoconvex polygon with k corners. There exists a nice pseudotriangle for P with cut $O(\log k)$.*

Proof. Observe first that there always exists *some* nice pseudotriangle ∇ for P. Select a corner w of P. Since P is strongly pseudoconvex, w "sees" a unique side chain of P. Together with w, the two corners that define this side chain span a nice pseudotriangle ∇. If ∇ has cut $O(\log k)$, then we are done. Otherwise, we apply induction by constructing a pseudotriangle whose cut is strictly smaller, as follows.

We will prove the following:

(*) at each edge d of ∇ which is a diagonal of P there exists an adjacent pseudotriangle ∇', spanned by d and some corner c of P, such that ∇' has cut at most $1 + \log_2 k$.

Property (*) is used as follows. Set $x = \log_2 k$, and assume that some side chain of ∇ contains at least $2x + 5$ diagonals of P. Choose a middlemost diagonal d, and generate an adjacent pseudotriangle ∇' as above. Flip edge d, which creates two pseudotriangles. Each of them avoids at least $x + 2$ diagonals that bound ∇ and contains at most $x + 1$ diagonals that bound ∇'. So for both, the cut is strictly

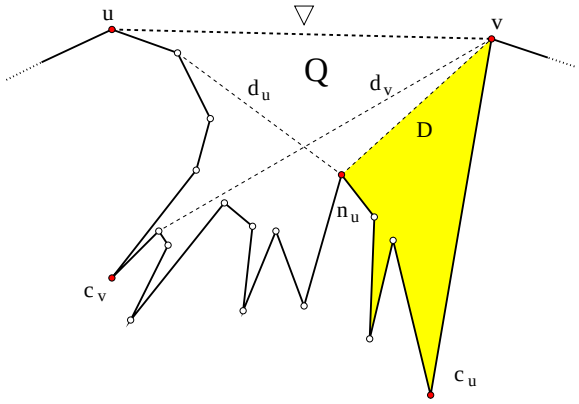


FIG. 4.1. *Constructing* ∇'.

smaller than the cut of ∇. Moreover, because P is strongly pseudoconvex, exactly one created pseudotriangle is nice.

It remains to prove (*). We construct the desired pseudotriangle ∇' as follows; see Figure 4.1. Let $d = uv$, and denote with Q the part of P cut off by d and not containing ∇. Note that u and v are corners of Q. Consider the minimum shortest-path tree of Q with root u, as defined in subsection 3.4. Let ∇_u be the unique face of this tree incident to uv. (Here and below, define things for v analogously.)

Case 1. Q contains only one corner $c \neq u, v$. Then $\nabla_u = \nabla_v$, and we take this face for the pseudotriangle ∇' in question. No diagonals of P lie on the side chains uc and vc of ∇'.

Case 2. Q contains at least two corners $\neq u, v$. Then ∇_u and ∇_v differ in their third corner, call it c_u and c_v, respectively, and thus have diagonals of P on their boundaries. Let d_u be the first diagonal on the side chain uc_u of ∇_u, and let d_u end at the vertex n_u. P is strongly pseudoconvex, and thus n_u is a noncorner of Q, because d_u would be a corner tangent, otherwise. Also, n_u does not lie on the side chain of Q that starts at v and is different from uv, because nonconvex geodesics inside P and via u and n_u would exist otherwise. Therefore, the geodesic from n_u to v starts with a diagonal D of P. Observe next that, by $\nabla_u \neq \nabla_v$, their side chains uc_u and vc_v cross. Hence the two diagonals d_u and d_v cross unless $n_u = n_v$. In both cases at least one diagonal, say d_u, cuts off from Q at least half of its corners. Therefore the diagonal D above, which is adjacent to d_u at vertex n_u, cuts off $k' \leq \frac{k}{2}$ corners. Clearly, $k' \geq 1$, and we treat the polygon cut off by D (shaded in Figure 4.1) recursively. The diagonal d_u lies on the boundary of the pseudotriangle ∇' being constructed.

Each occurrence of Case 2 increments the cut of ∇', which initially is 1 because of the diagonal $d = uv$. By the choice of D, Case 2 occurs at most $\log_2 k$ times. This proves (*) and completes the argumentation. ∎

4.2. Pseudotriangulations with small cut. We are now ready to prove the following structural result for minimum pseudotriangulations of simple polygons.

THEOREM 4.2. *For every polygon P with n vertices there exists a minimum pseudotriangulation of P in which each face has cut $O(\log n)$.*

Proof. We first partition P into strongly pseudoconvex polygons. Diagonals on nonconvex geodesics and corner tangents are used, such that each introduced diagonal

is incident to some corner in both polygons that it bounds. (These diagonals will contribute to the cut of the final faces in the minimum pseudotriangulation to be constructed, but their number is at most six per face.) Each strongly pseudoconvex polygon Q with more than three corners is partitioned further as follows.

Integrate a nice pseudotriangle ∇ with small cut for Q, whose existence is guaranteed by Lemma 4.1. Because ∇ is nice, it does not violate the pointedness of any vertex. Moreover, each diagonal of Q on ∇'s boundary is incident to two corners of the polygon it cuts off from Q. These polygons are partitioned recursively.

A minimum pseudotriangulation $\mathcal{MPT}$ of P results. Each face f of $\mathcal{MPT}$ has cut $O(\log n)$: A diagonal on f's boundary comes from Lemma 4.1 or is among the at most six edges incident to some corner of f. □

Remarks. Theorem 4.2 is asymptotically optimal. There exist polygons with n vertices where every possible minimum pseudotriangulation contains some face with cut $\Omega(\log n)$; see [6]. The theorem is related to a result in [35] which shows, for every point set S, the existence of a minimum pseudotriangulation with constant face complexity. Another related result, in [18], shows that every n-gon P admits a minimum pseudotriangulation $\mathcal{MPT}$ such that each line segment internal to P crosses only $O(\log n)$ edges of $\mathcal{MPT}$.

4.3. Splitting pseudotriangles. We continue with a ham-sandwich-type result for pseudotriangles.

LEMMA 4.3. *Let ∇ be a pseudotriangle that contains a set M of i points in its interior. There exists a point $p \in M$ whose geodesics to two corners of ∇ divide M into two subsets of cardinality $\frac{2\,i}{3}$ or less.*

Proof. For each point $p \in M$, the geodesics from p to the three corners of ∇ partition ∇ into three pseudotriangles (faces). Such a face f is called *sparse* if f encloses at most $\frac{2\,i}{3}$ points of M. We claim that, for each pair c, c' of corners of ∇, there exist at least $\frac{2\,i}{3} + 1$ sparse faces: Consider the sorted order of M, as given by the shortest-path tree with root c for M. The jth point of M in this order spans a face that contains strictly fewer than j points.

We conclude that there are at least $2\,i + 3$ sparse faces in total. Thus the mean number of sparse faces per point in M exceeds two, which implies that there exists a point $p \in M$ incident to three sparse faces. Among them, let f be the face that contains the most points, which are at least $\frac{i}{3}$. We take the two geodesics that span f to partition ∇. This yields two parts with at most $\frac{2\,i}{3}$ points each. □

Remarks. The fraction $\frac{2}{3}$ in Lemma 4.3 is optimal, even when ∇ is a triangle. The set M may consist of three groups of $\frac{i}{3}$ points such that, for each choice of $p \in M$, the two groups not containing p end up in the same subset.

4.4. Partition theorem. The results in the preceding subsections combine into the following partition theorem for augmented polygons.

THEOREM 4.4. *Let (P, S) be an augmented polygon, and let $I = S \setminus vert(P)$. There exist polygonal partitions Π_1 and Π_2 of (P, S) such that*

(1) *Π_1 uses $O(1)$ line segments spanned by S and assigns at most $\frac{2}{3} \cdot |I|$ points of I to each polygon;*

(2) *Π_2 uses $O(\log n)$ line segments spanned by S, assigns at most $\frac{2}{3} \cdot |I|$ points of I to each polygon, and guarantees the pointedness of each vertex of S.*

Proof. To construct Π_1, let $\mathcal{T}$ be some triangulation of the polygon P. Call a polygon $Q \subset P$ *sparse* if Q contains at most $\frac{2}{3} \cdot |I|$ points of I. Let ∇ be any face of $\mathcal{T}$. If each part of $P \setminus \nabla$ is sparse, then we are done, because we can partition P

with ∇, and ∇ with two line segments as in Lemma 4.3 if ∇ is nonsparse. Otherwise, we continue with the face of $\mathcal{T}$ adjacent to ∇ in the (unique) nonsparse part of $P \setminus \nabla$ until the first condition is met.

To construct Π_2 we proceed analogously but use a minimum pseudotriangulation $\mathcal{MPT}$ of P with face cuts bounded by $O(\log n)$. The existence of $\mathcal{MPT}$ is given by Theorem 4.2. The $O(\log n)$ edges of ∇ that are used to partition (P, S) retain the pointedness of all vertices, as do the two segments from Lemma 4.3 that may have to be used to split ∇. □

Remarks. Theorem 4.4 is similar in flavor to a result in [17], which asserts that any simple n-gon can be split by a diagonal into two subpolygons with at most $\frac{2}{3}n$ vertices.

5. Surfaces for pseudotriangulations. A *polyhedral surface* is the graph of a continuous and piecewise-linear function whose domain is the plane, or a polygonal subset thereof. Polyhedral surfaces are also known as polyhedral terrains. A two-dimensional cell complex $\mathcal{C}$ is called *projective* if there exists some polyhedral surface whose set of nonlinearity projects exactly to the set of all (internal) edges of $\mathcal{C}$. A subclass consists of the *regular* complexes, which are the projections of *convex* surfaces, i.e., whose defining function is convex. Examples of regular two-dimensional complexes are line arrangements, Voronoi diagrams and power diagrams (even their generalizations to higher order), and Delaunay triangulations; see, e.g., [9] and references therein. Not every triangulation, however, is regular (as Figure 7.3 below shows), but it is trivial that triangulations are projective.

We intend to show, in subsection 5.2, that pseudotriangulations have natural realizations as polyhedral surfaces. We characterize the class of projective pseudotriangulations in subsection 5.3 and discuss regular pseudotriangulations and their constrained variants in later sections.

The underlying domain is—as in previous sections—an augmented polygon, not only to gain generality compared to the point set case, but also because certain questions concerning a three-dimensional realization become more natural and interesting in this setting.

5.1. Complete vertices. We start with introducing a *status* for vertices, which will allow us to determine which vertices can be fixed in height when lifting a given pseudotriangulation to three dimensions.

Consider an augmented polygon (P, S), and let $\mathcal{PT}$ be a pseudotriangulation of (P, S). The status of a vertex v of $\mathcal{PT}$ is *complete* if v is a corner in each of its incident pseudotriangles. The status of v is *incomplete* otherwise. Note that each incomplete vertex uniquely corresponds to a pseudotriangle of $\mathcal{PT}$ where this vertex is a noncorner. Incomplete vertices have to be pointed, whereas complete vertices are pointed if and only if they are corners of P. In a (full) triangulation, all vertices are complete. On the other end of the spectrum, every minimum pseudotriangulation of (P, S) realizes the minimum number of complete vertices.

Observe that no vertex status is affected by an exchanging flip, whereas an edge-removing flip alters the status of exactly one vertex from complete to incomplete. A vertex-removing flip does not alter the status of any remaining vertex. Making all vertices complete by edge-inserting flips always results in a triangulation.

Remarks. We believe that the *completeness* of a vertex is a concept more natural than *pointedness*. For example, the vertices of $conv(S)$, though being pointed, behave like nonpointed vertices in the flipping process, because no geodesic ever runs via them. This effect becomes even more apparent when the underlying domain (P, S) is not convex. Corners as well as noncorners of the boundary polygon P may be pointed

in a pseudotriangulation $\mathcal{PT}$ of (P, S), but the (forced) pointedness of P's corners seems artificial. What actually counts is "(non)pointedness with boundary effects eliminated," that is, the completeness status of a vertex.

The number of incomplete vertices is $i = p - k$ if P has k corners and $\mathcal{PT}$ has p pointed vertices. This relation saves one of the four parameters in the formulas for the number of edges and faces in a pseudotriangulation of (P, S) (see subsection 3.2), another fact in favor of the new concept.

Complete vertices can be distinguished from incomplete ones in any polygonal partition, as they are defined via corners of polygonal faces. In fact, all the results in sections 2, 3, and 4 could have been derived using this terminology. We refrained from doing so only to keep to existing notation in related work. In the three-dimensional setting below, the notion of complete vertices is indispensable, however.

5.2. Surface theorem. In this subsection, we state and prove a main geometric result of this paper.

THEOREM 5.1 (surface theorem). *Let (P, S) be an augmented polygon, and let $\mathcal{PT}$ be any pseudotriangulation thereof. Let $\boldsymbol{h}$ be a vector assigning a height to each complete vertex of $\mathcal{PT}$. For each choice of $\boldsymbol{h}$ there exists a unique polyhedral surface $\mathcal{F}$ above the domain P, which respects $\boldsymbol{h}$ and whose edges project vertically to (a subset of) the edges of $\mathcal{PT}$.*

Proof. Let $v_1, \ldots, v_r$ denote the complete vertices of $\mathcal{PT}$, and let $v_{r+1}, \ldots, v_n$ be the incomplete ones. We set up a system of linear equations for the vertex heights $z_1, \ldots, z_n$ that determines a polygonal surface $\mathcal{F}$ and has a unique solution.

If v_i is a complete vertex, the system contains the equation $z_i = h_i$. Otherwise, its height z_i is constrained as follows. There is a unique pseudotriangle ∇ in $\mathcal{PT}$ where v_i is no corner; an internal angle larger than π occurs in ∇ at v_i. Let v_j, v_k, v_ℓ be the corners of ∇. We have $v_i \in \mathit{conv}(v_j, v_k, v_\ell) = \mathit{conv}(\nabla)$ by definition of a pseudotriangle. Requiring coplanarity of the lifted vertices v_i, v_j, v_k, v_ℓ leads to an equation of the form

$$\alpha(v_j v_k v_\ell) \cdot z_i - \alpha(v_i v_k v_\ell) \cdot z_j - \alpha(v_i v_j v_\ell) \cdot z_k - \alpha(v_i v_j v_k) \cdot z_\ell = 0,$$

where $\alpha(abc)$ denotes the (unsigned) area of a triangle abc. This expresses z_i as a convex combination of z_j, z_k, and z_ℓ.

A linear system $A \cdot \boldsymbol{z} = \boldsymbol{b}$ is obtained, where A is an $n \times n$ matrix whose ith row corresponds to the equation for z_i, and where $\boldsymbol{b}$ is a vector that coincides with $\boldsymbol{h}$ in the first r entries and contains zero entries otherwise. Any solution $\boldsymbol{z}$ defines a valid surface $\mathcal{F}$, because triangular facets of $\mathcal{F}$ are trivially fixed by three heights, and for each pseudotriangular facet of $\mathcal{F}$ its vertices are forced to be coplanar. Thus $\mathcal{F}$ is piecewise-linear and continuous, and each edge of $\mathcal{F}$ projects to an edge of $\mathcal{PT}$.

We now prove that a unique solution $\boldsymbol{z}$ exists, by showing $\mathit{det}(A) \neq 0$. Call an $n \times n$ matrix M *affirmative* if (a) M is positive exactly at its diagonal entries, (b) no row sum in M is negative, and (c) at least one row sum is positive. M is called *strictly affirmative* if each allowable submatrix of M (i.e., which comes from deleting the ith row and the ith column of M for $t < n$ indices i) is affirmative. It is easy to prove $\mathit{det}(M) > 0$ for any strictly affirmative matrix M; see [10].

It remains to show that A is strictly affirmative. Observe first that A is affirmative: The first $r \geq 1$ rows of A are unit rows, and the remaining $n - r$ rows represent convex combinations for the diagonal elements. Let A' be an allowable submatrix of A. Then properties (a) and (b) clearly hold for A'. Property (c) is obvious while unit rows (that correspond to complete vertices) are present in A', and is retained because heights

for complete vertices participate with negative coefficients in remaining rows. We conclude that A is strictly affirmative. □

Remarks. The set of edges of the surface $\mathcal{F}$ in Theorem 5.1 may project to a proper subset of edges of the pseudotriangulation $\mathcal{PT}$ of (P, S). We will discuss this phenomenon in detail in subsection 5.3.

The existence of *some* projection surface is obvious when (P, S) contains no internal vertices, that is, if $S = vert(P)$. Then all internal edges of $\mathcal{PT}$ are diagonals of P. For each pseudotriangle of $\mathcal{PT}$, its three corners define a plane, which can be placed appropriately in space. Note that these planes can be chosen such that the resulting surface is convex at each internal edge; this observation leads to Corollary 9.2 in section 9.

Theorem 5.1 holds for arbitrary (augmented) polygonal domains, including polygons with holes. We will use this generalization in subsections 6.3 and 8.2. Note that the noncorners of holes are corners of the domain and thus are complete vertices.

The vector $\boldsymbol{b}$ is an eigenvector of the transpose A^T with eigenvalue 1, that is, $A^T \cdot \boldsymbol{b} = \boldsymbol{b}$. In fact, A^T (and thus A) has r eigenvalues of 1.

Interestingly, a matrix of a form similar to A arises in the context of power diagrams and Gale transforms [10]. Moreover, the polytope representation of pseudotriangulations, which we are going to describe in section 9, could alternatively be obtained using Gale transform techniques; cf. [15]. A systematic use of Gale transforms may shed additional light on the geometric properties of pseudotriangulations.

By applying allowable row operations to A, it can be shown that its inverse A^{-1} contains only nonnegative entries. This implies a monotonicity property which will be important later.

LEMMA 5.2. *The solution $\boldsymbol{z}$ of the system $A \cdot \boldsymbol{z} = \boldsymbol{b}$ is a monotone (linear) function of $\boldsymbol{b}$ and thus of the height vector $\boldsymbol{h}$.*

The Maxwell–Cremona theorem [20] asserts that, for a given planar straight-line graph G, there exists a (certain) projection surface if and only if G admits a so-called stress, i.e., an assignment of a real-valued tension to each edge of G such that all vertices of G are in equilibrium state. In this projection surface, the outer facet is required to be planar (unlike in the surface in Theorem 5.1). For *convex* augmented polygons (P, S), the Maxwell–Cremona theorem combines with Theorem 5.1, namely, when height zero is chosen for each (necessarily complete) boundary vertex of the convex polygon P. This gives the corollary below, which also follows from recent results in [41].

COROLLARY 5.3. *Let $\mathcal{PT}$ be a pseudotriangulation of a planar point set S. There exists a nontrivial stress for the edges of $\mathcal{PT}$, provided that $\mathcal{PT}$ is not minimum. More generally, the dimension of the space of stresses for $\mathcal{PT}$ equals the number of its complete internal vertices.*

We mention that Theorem 5.1 also implies a variant of Corollary 5.3 for arbitrary augmented polygons (P, S), asserting that every pseudotriangulation of (P, S) admits a nontrivial stress for the set $S \setminus vert(P)$ of *internal* vertices unless P is a pseudotriangle.

5.3. Projective pseudotriangulations. The surface $\mathcal{F}$ in Theorem 5.1 may fail to be strictly convex or reflex for some edges of $\mathcal{PT}$. In this case, $\mathcal{F}$ yields a proper subset of edges of $\mathcal{PT}$ in the projection. One reason for this is that the height vector $\boldsymbol{h}$ is degenerate. For instance, if $\boldsymbol{h} = 0$, then $\mathcal{F}$ contains a single facet that projects to the polygon P in the plane, regardless of the shape of $\mathcal{PT}$. On the other hand, pseudotriangles in double-adjacency give rise to a single pseudotriangular facet

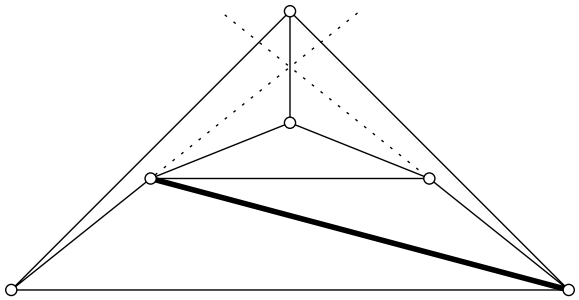


FIG. 5.1. *A stable though nonprojective pseudotriangulation.*

of $\mathcal{F}$ for every choice of $\boldsymbol{h}$.

Let us call a pseudotriangulation $\mathcal{PT}$ of (P, S) *stable* if no subset of incomplete vertices in $S \setminus vert(P)$ can be eliminated (along with their incident edges) such that both a valid pseudotriangulation $\mathcal{PT}'$ remains, and the status of each vertex of $\mathcal{PT}'$ is the same as in $\mathcal{PT}$. In particular, double-adjacencies are ruled out in a stable pseudotriangulation. Trivial stable instances are full triangulations, and pseudotriangulations of (P, S) for $S = vert(P)$. To see an example of a pseudotriangulation which is not stable, omit the three bold edges in Figure 7.3 (below) which do not belong to the internal triangle. Interestingly, no double-adjacencies occur in this example.

THEOREM 5.4. *A pseudotriangulation $\mathcal{PT}$ of (P, S) is projective only if $\mathcal{PT}$ is stable. If $\mathcal{PT}$ is stable, then the point set S can be perturbed (by some arbitrarily small ε) such that $\mathcal{PT}$ becomes projective.*

Proof. Assume first that $\mathcal{PT}$ is not stable. Eliminate some set of incomplete vertices from $S \setminus vert(P)$ such that a pseudotriangulation $\mathcal{PT}'$ of (P, S) is obtained where all vertices agree in status with their counterparts in $\mathcal{PT}$. Fix an arbitrary height vector $\boldsymbol{h}$ for S. By Theorem 5.1, there exist unique surfaces $\mathcal{F}$ for $\mathcal{PT}$ and $\mathcal{F}'$ for $\mathcal{PT}'$. Since only incomplete vertices have been eliminated, and the status of no remaining vertex has been changed, we have $\mathcal{F}' = \mathcal{F}$. Now let ∇ be a pseudotriangle of $\mathcal{PT}'$ but not of $\mathcal{PT}$. The restriction of $\mathcal{F}'$ to ∇ is a pseudotriangular facet. Because $\mathcal{F}' = \mathcal{F}$, the restriction of $\mathcal{F}$ to ∇ is planar, too. This implies that $\mathcal{PT}$ is not projective.

Now let $\mathcal{PT}$ be a stable pseudotriangulation. Let $\boldsymbol{h}$ be a height vector for S with pairwise different entries. Assume that the surface $\mathcal{F}$ for $\mathcal{PT}$ and $\boldsymbol{h}$ is planar at some edge uv. Let c and c' be the two corners opposite to uv in the two pseudotriangular facets of $\mathcal{F}$ adjacent at uv. By assumption, the vertex v lies in the plane through u, c, and c'. Suppose first that v is a complete vertex. As the heights of u, c, and c' also depend on vertices different from v, we can displace v by some arbitrarily small ε in an appropriate direction, such that $\mathcal{F}$ becomes nonplanar at uv. Now suppose that v is incomplete. Because $\mathcal{PT}$ is stable, the height of v does not depend on the same set of (three) vertices as do the heights of u, c, and c'. Otherwise, v could be eliminated (possibly jointly with other vertices) without changing $\mathcal{F}$, a contradiction to the assumed stability of $\mathcal{PT}$. So, again, v can be displaced to make $\mathcal{F}$ nonplanar at uv. This implies that S can be perturbed such that $\mathcal{PT}$ becomes projective. □

Figure 5.1 illustrates a stable pseudotriangulation which fails to be projective. Neither of the two incomplete vertices can be eliminated without changing the status of the upmost internal vertex from complete to incomplete. On the other hand, as the prolongations (shown dotted) of two edges meet at a third edge, the edge drawn

in bold flattens out in all possible surface realizations. Note that a slight movement of any single vertex restores the projectivity in this example.

By Theorem 5.4, we restrict our attention—in the remainder of this paper—to vertex sets where every stable pseudotriangulation is also projective. For such a vertex set S and a pseudotriangulation $\mathcal{PT}$ thereof, a height vector $\boldsymbol{h}$ is called *generic* if $\boldsymbol{h}$ witnesses the projectivity of $\mathcal{PT}$.

Remarks. Note the difference between the concepts of *stable* and *minimum* pseudotriangulation. In the latter, no single edge (but possibly some vertices) can be eliminated such that a valid pseudotriangulation remains. There is no obvious relation between stable and minimum pseudotriangulations, but the following can be observed. If $conv(S)$ is a triangle and S contains vertices internal to P, then no minimum pseudotriangulation $\mathcal{MPT}$ of (P, S) is stable. All internal vertices are incomplete, and their elimination leaves a valid pseudotriangle P without changing the status of any vertex of P. By Theorem 5.4, this implies that $\mathcal{MPT}$ is not projective.

6. Surface interpretation of flips. We next provide a surface interpretation for all the admissible flip operations in section 2. To this end, consider some projective pseudotriangulation $\mathcal{PT}$ of an augmented polygon (P, S). (Recall from subsection 5.3 that "projective" is equivalent to "stable," by our convention on the vertex set S.) Let $\mathcal{PT}'$ be the unique pseudotriangulation obtained from $\mathcal{PT}$ by applying a single admissible flip. Assign height vectors $\boldsymbol{h}$ and $\boldsymbol{h}'$ to $\mathcal{PT}$ and $\mathcal{PT}'$ such that $\boldsymbol{h}$ is generic for $\mathcal{PT}$ and coincides with $\boldsymbol{h}'$ for all vertices being complete in both structures. By Theorem 5.1, there exist two unique projection surfaces $\mathcal{F}$ and $\mathcal{F}'$ for $\mathcal{PT}$ and $\mathcal{PT}'$. We call the operation that transforms $\mathcal{F}$ into $\mathcal{F}'$ a *surface flip*.

In a combinatorial sense, surface flips cause local and constant-size changes. Geometrically, however, these operations are by no means local. A single flip may change the surface heights for many vertices, even "far away" from the region where the combinatorial change takes place. The change in height depends on the structure of the entire surface. Thus general surface flips are *context-sensitive* operations.

For expository reasons, we will first study surface flips in their *context-free* version in the next two subsections. The following notation will be used. Consider an exchanging or an edge-removing flip. ∇_1 and ∇_2 are the two adjacent pseudotriangles involved in the flip, and R denotes their union. We restrict the corresponding surface flip to R such that $\mathcal{PT}$ and $\mathcal{PT}'$ are the two (projective) pseudotriangulations of R before and after the flip, and $\mathcal{F}$ and $\mathcal{F}'$ are the corresponding unique surfaces. The operation that takes $\mathcal{F}$ into $\mathcal{F}'$ constitutes the context-free surface flip. We come back to its context-sensitive version in subsection 6.3.

6.1. Convexifying flips. Suppose the flip in question is an exchanging flip. The following combinatorial change is caused. The flip replaces the edge $e = \nabla_1 \cap \nabla_2$ by the edge e' that is the intersection of the two pseudotriangles that comprise $\mathcal{PT}'$. Note that ∇_1 and ∇_2 are in no double-adjacency, by our assumption that $\mathcal{PT}$ is projective. R has exactly four corners, which are the vertices v_i, complete in both $\mathcal{PT}$ and $\mathcal{PT}'$. Their heights have been fixed as $z_i = h_i$, in a noncoplanar manner. The noncorners of R are incomplete in both of $\mathcal{PT}$ and $\mathcal{PT}'$. If they change incidence to pseudotriangles during the flip, then the planarity conditions for their heights are affected. For such a vertex v_j the change is from $z_j = \gamma_j$ to $z_j = \gamma'_j$, where γ_j and γ'_j are convex combinations of the heights of the corners in the respective pseudotriangles.

To view the change geometrically, see Figure 6.1. Consider the two facets f_1 and f_2 of the surface $\mathcal{F}$ before the flip, and assume that $\mathcal{F}$ is reflex at edge e. Then exactly one corner of f_1 lies below the plane through f_2, and vice versa. As corners

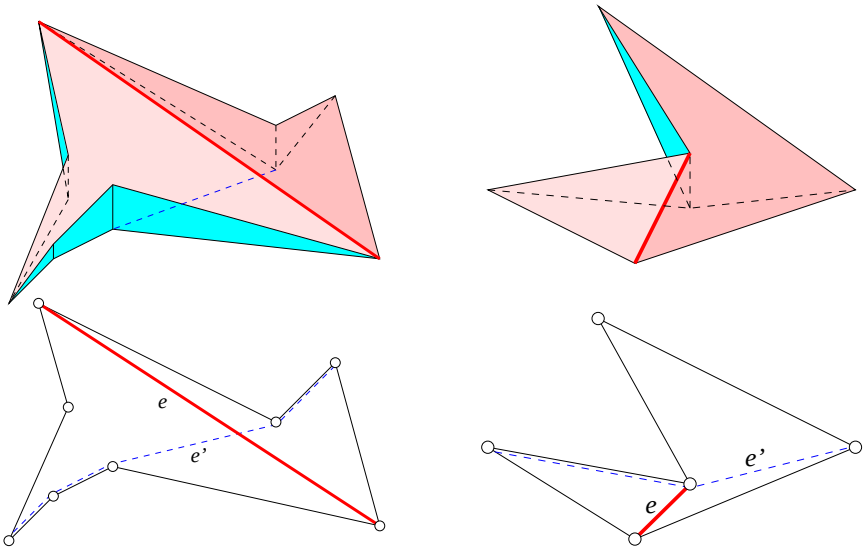


FIG. 6.1. *Two convexifying flips.*

do not change height in the flip, this implies that the obtained surface $\mathcal{F}'$ is convex at edge e'. We will therefore use the term *convexifying* flip. $\mathcal{F}$ and $\mathcal{F}'$ bound a polyhedron from above and below, respectively. This polyhedron is a tetrahedron if f_1 and f_2 are triangles, and the flip degenerates to the Lawson flip for triangulations. In this case, we will also talk of a *tetrahedral* flip—a special case well known from flipping in triangular surfaces; see, e.g., [38].

6.2. Planarizing flips. We discuss the edge-removing flip next. Combinatorially, the following change takes place. $\mathcal{PT}$ consists of two pseudotriangles ∇_1 and ∇_2 in single adjacency. The flip removes their common edge e and joins ∇_1 and ∇_2 to the pseudotriangle R, which thus represents $\mathcal{PT}'$. The three corners of R are complete in both $\mathcal{PT}$ and $\mathcal{PT}'$. Thus their heights are given explicitly. One vertex v_e incident to e changes its status from complete in $\mathcal{PT}$ to incomplete in $\mathcal{PT}'$. The condition for its height changes from $z_e = h_e$ to $z_e = \gamma'_e$, where γ'_e expresses coplanarity with the lifted corners of R. All other vertices v_j of R are incomplete in both $\mathcal{PT}$ and $\mathcal{PT}'$. Their height constraints change from $z_j = \gamma_j$ to $z_j = \gamma'_j$, where γ_j expresses coplanarity with the lifted corners of ∇_1 or ∇_2.

Geometrically, the surface $\mathcal{F}$ (which we assume to be reflex at edge e again) is flipped to $\mathcal{F}'$, as is depicted in Figure 6.2. $\mathcal{F}'$ is a single pseudotriangular facet and thus is planar. We call such a flip a *planarizing* flip. Again, $\mathcal{F}$ and $\mathcal{F}'$ bound a polyhedron from above and below, respectively.

6.3. Context-sensitive view. The context-free flips described in the previous two subsections leave the heights of the complete vertices of $\mathcal{PT}'$ unchanged. The incomplete vertices of $\mathcal{PT}'$ are lowered. Complete vertices may, however, be incomplete when being embedded in a larger pseudotriangulation. A surface flip—in its general context-sensitive form—may alter their heights as well. This alteration is uniquely described by the corresponding two linear systems. No alteration occurs in a convexifying flip that is tetrahedral, because the system remains unchanged.

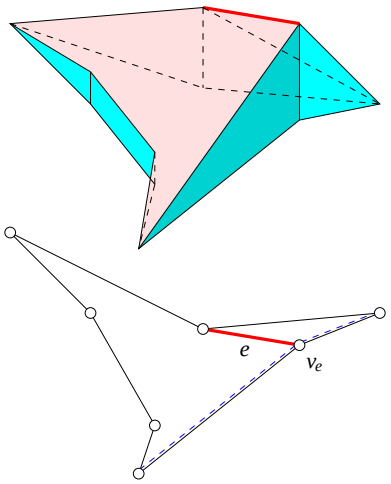


FIG. 6.2. *Planarizing flip.*

LEMMA 6.1. *Let $\mathcal{F}'$ be obtained from the surface $\mathcal{F}$ by applying a (context-sensitive) convexifying or planarizing surface flip φ. Then $\mathcal{F}'$ nowhere lies strictly above $\mathcal{F}$.*

Proof. We show that φ can be simulated by a sequence of flips, each lowering certain parts of the surface. Let (P, S) and $\mathcal{PT}$, respectively, be the augmented polygon and the pseudotriangulation in question. Thus $\mathcal{F}$ is the surface for $\mathcal{PT}$ before flipping. Consider the polygon $R \subseteq P$ to which the context-free version of φ is restricted, and denote this flip by φ_R.

Introduce new edges to $\mathcal{PT}$ such that a pseudotriangulation $\mathcal{PT}_1$ is obtained in which all corners of R are complete vertices. Assign heights as determined by $\mathcal{F}$ to the complete vertices of $\mathcal{PT}_1$. The surface that corresponds to $\mathcal{PT}_1$ and these heights is still $\mathcal{F}$. Now perform the flip φ_R and denote the obtained pseudotriangulation by $\mathcal{PT}_2$ and its surface by $\mathcal{F}_2$. This lowers the noncorners of R (which are incomplete in $\mathcal{PT}_2$) but leaves the heights of R's corners fixed. To see that no other incomplete vertex of $\mathcal{PT}_2$ can gain height, let us cut out the region R from P and consider it as a hole. The restrictions of $\mathcal{PT}_1$ and $\mathcal{PT}_2$, respectively, to $P \setminus R$ have the same set of complete vertices, which now includes all vertices of R. Let $\boldsymbol{h}_1$ and $\boldsymbol{h}_2$ be their height vectors in the linear systems for $\mathcal{F}$ and $\mathcal{F}_2$. We know that $\boldsymbol{h}_2 \leq \boldsymbol{h}_1$ (element-wise) because a subset of R's vertices has been lowered, which implies $\boldsymbol{z}_2 \leq \boldsymbol{z}_1$ by Lemma 5.2. We conclude that $\mathcal{F}_2$ nowhere lies strictly above $\mathcal{F}$.

Observe that pseudotriangles in $\mathcal{PT}_1$ may be incident to R only with their corners. Thus $\mathcal{F}_2$ may be reflex at edges of $\mathcal{PT}_1$ which are not in $\mathcal{PT}$. However, no reflex edge is changed to being convex. If all reflex edges of $\mathcal{F}_2$ have been present as reflex edges in $\mathcal{F}$ already, then the surface flip is complete. That is, $\varphi_R = \varphi$. Otherwise, appropriate context-free flips (convexifying or planarizing) are applied to the new reflex edges in $\mathcal{F}_2$, in the way described above and until all such edges disappear. Note that reflex edges flipped away earlier can reappear, so that the process may oscillate. It converges, though, because each reflex edge is flippable at all times, and there is an obvious lower bound on the heights, as certain vertices (the corners of P, for instance) never change height. This implies that the limit is the unique surface $\mathcal{F}'$ that $\mathcal{F}$ is flipped into by the context-sensitive surface flip φ. □

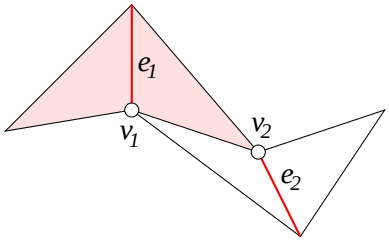


FIG. 6.3. *Oscillating flips.*

Figure 6.3 illustrates two oscillating flips. The edges e_1 and e_2 are both reflex at the beginning. They alternately disappear when context-free planarizing flips are applied. Their endpoints v_1 and v_2 are lowered alternately until the limit is reached, the pseudotriangulation that contains the shaded and the blank pseudotriangles.

6.4. Trivial flips. In a convexifying or a planarizing surface flip, a pseudotriangulation of (P, S) may be created which is not stable. This happens if a subset $B \in S \setminus vert(P)$ of incomplete vertices is generated, such that the removal of B leaves a valid pseudotriangle ∇ without changing the status of vertices. As a special case, a double-adjacency occurs. The obtained pseudotriangulation is not projective, by Theorem 5.4. In fact, the surface flip changes the part of the surface above ∇ to planar. That is, a *trivial* flip takes place implicitly, which eliminates from the surface the edges and vertices that stem from B.

In the pseudotriangulation of (P, S), the removal of B can be realized by a finite sequence of admissible flips (including vertex-removing flips, of course). This restores a projective pseudotriangulation. Surface flips that cause trivial flips are a natural generalization of the degree-3 vertex removal in triangular surfaces, where a single tetrahedron is split off; see, e.g., [38].

Remarks. Each (nontetrahedral) surface flip requires that a system of linear equations be solved; see subsection 5.2. The corresponding matrix has a special structure, however. The first $n-i$ rows are unit rows if i of the n vertices in the current pseudotriangulation are incomplete. Thus the system can be solved in $O(n+i^3)$ time, which covers the geometrical part of the surface flip. The combinatorial part obviously can be done within this complexity.

7. Locally convex functions. Let P be a polygon in the plane. A real-valued function f with domain P is called *locally convex* if f is convex on each line segment internal to P. For convex polygons P, every locally convex function is convex continuable on the exterior of P, but this is not true for arbitrary polygons in general.

Let (P, S) be an augmented polygon, and let $\boldsymbol{h}$ be a vector assigning some real value to each of the n vertices in S. We are interested in the following optimization problem.

Given (P, S) and $\boldsymbol{h}$, find the maximal locally convex function f^* on the domain P which fulfills $f^*(v_i) \leq h_i$ for each $v_i \in S$.

The function f^* is unique, because it is the pointwise maximum of all locally convex functions on P that satisfy the constraints given by $\boldsymbol{h}$. Moreover, f^* is piecewise linear, and nonlinearity occurs only at line segments spanned by the set S. We prove that the graph $\mathcal{F}^*$ of f^* projects to a pseudotriangulation of (P, S') for some $S' \subseteq S$. We further show that $\mathcal{F}^*$ can be constructed by a finite sequence of convexifying or planarizing surface flips.

If (P, S) is convex, then a well-known special case is obtained: $\mathcal{F}^*$ is the lower convex hull of the point set S when being lifted to the heights in $\boldsymbol{h}$. In this case, we succeed in generating the (convex) surface $\mathcal{F}^*$ with $O(n^2)$ surface flips of the kind above. The bound $O(n^2)$ can further be achieved when (P, S) is an arbitrary polygon without internal vertices. Both flip distance results are asymptotically optimal, by known lower bounds; see, e.g., [26].

7.1. Flipping to optimality. Consider some full triangulation $\mathcal{T}$ of the augmented polygon (P, S). All the n vertices of $\mathcal{T}$ are complete vertices, and we assign to $\mathcal{T}$ the height vector $\boldsymbol{h}$ used above to specify the maximal locally convex surface $\mathcal{F}^*$. This defines a unique triangular surface $\mathcal{F}(\mathcal{T}, \boldsymbol{h})$. We state another core result of this paper.

THEOREM 7.1 (optimality theorem). *Let $\mathcal{F}^*(\mathcal{T}, \boldsymbol{h})$ be a surface obtained from $\mathcal{F}(\mathcal{T}, \boldsymbol{h})$ by applying convexifying and planarizing surface flips (in any order) as long as reflex edges do exist. Then $\mathcal{F}^*(\mathcal{T}, \boldsymbol{h}) = \mathcal{F}^*$ for any choice of the initial triangulation $\mathcal{T}$. The optimum $\mathcal{F}^*$ is reached after a finite number of surface flips.*

Proof. By Lemma 6.1, convexifying and planarizing surface flips always generate a sequence of surfaces that is (strictly) totally ordered by decreasing height. As, by the uniqueness result in Theorem 5.1, these surfaces are pairwise different, so are the corresponding pseudotriangulations. An exponential upper bound on the sequence length results, because the number of possible planar straight-line graphs on the vertex set S is exponentially bounded in n; see [7]. The obtained surface $\mathcal{F}^*(\mathcal{T}, \boldsymbol{h})$ contains no reflex edge, as each such edge can be flipped at any time. This implies that $\mathcal{F}^*(\mathcal{T}, \boldsymbol{h})$ is locally convex. Moreover, $\mathcal{F}^*(\mathcal{T}, \boldsymbol{h})$ is maximal, because it fulfills $\boldsymbol{h}$ with equality for all complete vertices. Thus $\mathcal{F}^*(\mathcal{T}, \boldsymbol{h}) = \mathcal{F}^*$ follows from the uniqueness of $\mathcal{F}^*$. □

If the vector $\boldsymbol{h}$ constraining the vertex heights is generic, then all facets in the surfaces generated above are pseudotriangles. That is, only (projective) pseudotriangulations are generated. In particular, $\mathcal{F}^*$ projects to a pseudotriangulation.

$\mathcal{F}^*$ can be alternatively computed by solving a linear program with at most $\binom{n}{4}$ constraints. Each constraint stems from a quadrilateral $Q \subset P$ with vertices from S and postulates convex position for the vertex heights.

Theorem 3.4 implies the existence of a sequence of $O(n \log n)$ surface flips that transforms any given triangular surface $\mathcal{F}(\mathcal{T}, \boldsymbol{h})$ for (P, S) into $\mathcal{F}^*$. This sequence includes convexifying and planarizing flips and, in general, their inverses. Trivial flips may be implicated if vertices of $\mathcal{F}(\mathcal{T}, \boldsymbol{h})$ are missing in $\mathcal{F}^*$. The sequence contains no inverse trivial flips, however. This rules out the reinsertion of vertices and makes the bound substantial. Note that the sequence need not be *improving*, i.e., exclusively consisting of convexifying and planarizing (and trivial) surface flips such as in Theorem 7.1. We summarize in the following.

THEOREM 7.2. *Let (P, S) be an augmented polygon with $|S| = n$, and let $\boldsymbol{h}$ be a height vector for S. Let f^* be the unique maximal locally convex function on P that is bounded from above by $\boldsymbol{h}$.*

(1) *The graph $\mathcal{F}^*$ of f^* projects to a pseudotriangulation in the generic case.*

(2) *$\mathcal{F}^*$ can be constructed from any triangular surface $\mathcal{F}$ for (P, S) and $\boldsymbol{h}$, by applying any improving flip sequence of sufficient length.*

(3) *There exists a sequence of $O(n \log n)$ flips (excluding the inverse trivial type) which transforms $\mathcal{F}$ into $\mathcal{F}^*$ and can be computed in polynomial time if $\mathcal{F}^*$ is known.*

A principal problem, which complicates the construction of improving sequences of polynomial length (as in [26]), is that edges flipped away may reappear. Figure 7.1

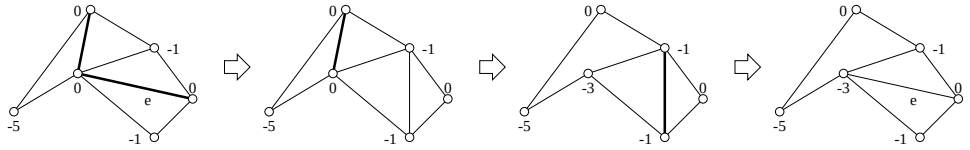


FIG. 7.1. *Edge e reappears.*

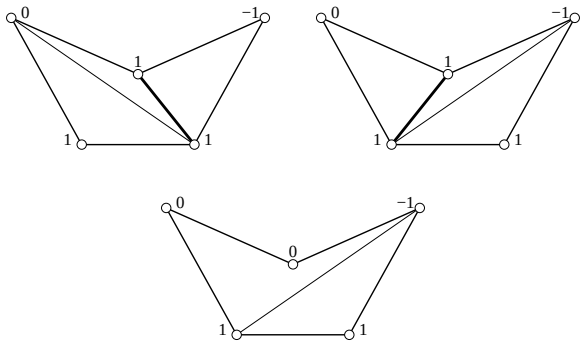


FIG. 7.2. *Local and global optima.*

illustrates this phenomenon. Numbers denote vertex heights, and reflex edges are shown in bold. Edge e is flipped first, and a planarizing flip for the other reflex edge follows. This makes the edge between the two remaining triangles reflex. Flipping this edge next lets e reappear, though as a convex edge. However, an appropriate flip in the neighborhood of e (not shown in the figure) makes e reflex again. Interestingly, a different improving sequence—that does not start with e—avoids this undesirable effect. We show in subsections 7.2 and 7.3 that the reappearance of edges can always be avoided for less general domains (P, S).

Remarks. Flipping to local convexity is not always possible *within* the class of triangulations. Different flip sequences may terminate at different triangular surfaces that contain reflex edges, as Figure 7.2 shows. (Numbers at vertices denote surface heights.) The initial surface contains two reflex edges, connecting vertices of height 1. Flipping either edge yields one of the two triangular surfaces shown in the upper part. Each contains only one reflex edge (shown in bold), which cannot be flipped away without destroying the triangulation. The global optimum $\mathcal{F}^*$ is a pseudotriangular surface, shown in the lower part.

7.2. Convex domains. Suppose that the underlying domain (P, S) is convex. Then $\mathcal{F}^*$ bounds from below the convex hull of a three-dimensional point set. Let us show that reappearance of edges does not take place if improving surface flips are performed in a carefully chosen order. In particular, the following holds.

LEMMA 7.3. *Let (P, S) be an augmented* convex *polygon, and let $\boldsymbol{h}$ be a height vector for S. Then $\mathcal{F}^*$ can be generated, from any given triangular surface for (P, S) and $\boldsymbol{h}$, by $O(n^2)$ improving surface flips, for $n = |S|$.*

Proof. To avoid the discussion of special cases, assume that $\boldsymbol{h}$ is generic. Let $\mathcal{F}$ be the current surface for (P, S), which is triangular at the beginning. For a vertex v of $\mathcal{F}$, denote with $I(v)$ the set of all vertices of $\mathcal{F}$ that share a facet with v. Note that $v \in I(v)$. Let $low(I(v))$ be the lower part of the boundary of $conv(I(v))$. We use the following algorithm to flip $\mathcal{F}$ into $\mathcal{F}^*$.

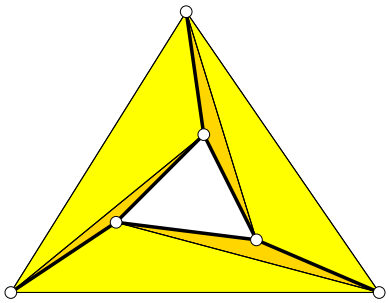


FIG. 7.3. *Local optimum for convex P; adapted from* [25].

Step 1. While there exists a vertex v of $\mathcal{F}$ that lies strictly above $low(I(v))$ do:

Case 1.1. Vertex v is complete. As v lies strictly above $low(I(v))$, there exist reflex edges incident to v. While v is complete, we flip away such edges, using convexifying and planarizing surface flips. We know, from Lemma 6.1, that $low(I(v))$ is lowered in this process, whereas v keeps its height. Thus there always exists some reflex edge incident to v, which can be flipped. Each flip decrements the degree of v. Note that pseudotriangular facets with many vertices may be created, but only if v is a corner there. Obviously, the last flip is planarizing. It either changes the status of v to incomplete, and we continue with Case 1.2, or it implicates a trivial flip that removes v (possibly together with some other vertices in v's neighborhood). In particular, v gets removed if its degree before the flip has been three.

Case 1.2. Vertex v is incomplete. Let f be the unique pseudotriangular facet of $\mathcal{F}$ where v is a noncorner. While v has more than two edges, we flip away reflex edges, according to the following rule. As v lies strictly above $low(I(v))$, either both edges of f incident to v are reflex, or another reflex edge is incident to v. In the latter case, we flip this edge. The former case needs more care. Consider the projections, e_1 and e_2, of these two reflex edges of f in the pseudotriangulation. Between them, there is another edge incident to the projection v' of v. Therefore, for at least one of e_1 and e_2, say e_1, the geodesic between the two corners opposite to e_1 (in the two pseudotriangles adjacent at e_1) does not run via v'. We select e_1 for the edge to be flipped. This flip creates no edge incident to v. Thus either the degree of v is decremented, or v is removed in a trivial flip.

Step 2. Now all vertices v of $\mathcal{F}$ lie on $low(I(v))$. The surface $\mathcal{F}$ therefore is triangular, and all vertices in the corresponding triangulation are complete. While $\mathcal{F}$ is not convex, we apply convexifying (in fact, tetrahedral) flips. No vertex heights change. This yields the convex and triangular surface $\mathcal{F}^*$.

The total number of flips in Cases 1.1 and 1.2 is bounded by $n-1$, the maximum possible degree of the vertex v. Step 1 runs through these cases at most $n-3$ times, because each time one internal vertex is removed. The number of flips needed in Step 2 is at most $\binom{n}{2}$, by well-known arguments [26]. □

Remarks. Even though P is convex such that $\mathcal{F}^*$ is triangular, only a local optimum may be reached when tetrahedral flips are applied exclusively. See Figure 7.3, which is taken from [25]. P has height zero, and the three vertices in $S \setminus vert(P)$ lie in some plane above P. No reflex edge (bold) can be flipped, and no single vertex can be removed, to obtain a valid triangulation. Planarizing (and trivial) flips are needed

to reach the optimum $\mathcal{F}^*$, which is a single triangle P.

It remains unclear whether the algorithm above can be modified to run efficiently for *nonconvex* augmented polygons (P, S). Step 1 may leave noncorners of P incomplete, such that Step 2 is not guaranteed to take $O(n^2)$ flips.

7.3. Vertex-empty domains. Improving flip sequences of (at most) quadratic length also exist for any polygonal domain (P, S) without internal vertices, that is, in the case $vert(P) = S$. Let Δ be an *ear* of the polygon P, that is, a triangle that can be cut off from P by some diagonal. It is well known that every triangulation of P contains at least two ears.

LEMMA 7.4. *Let Δ be an ear of P, and let $\boldsymbol{h}$ be a height vector for $vert(P)$. Denote with $\mathcal{F}^*$ and $\mathcal{F}^*(P \setminus \Delta)$, respectively, the optimal surfaces for P and $P \setminus \Delta$ with respect to $\boldsymbol{h}$. Then $\mathcal{F}^*(P \setminus \Delta)$ can be transformed into $\mathcal{F}^*$ by attaching the triangular facet for Δ and applying $O(n)$ improving flips.*

Proof. Consider the following sequence of flips. Imagine that a triangular facet f with the following properties is attached to $\mathcal{F}^*(P \setminus \Delta)$: The projection of f is Δ, and the corner c of f that does not belong to $\mathcal{F}^*(P \setminus \Delta)$ has some height h_c large enough to make the resulting surface convex at the respective diagonal d of P. This surface is optimum for P and the vector $\boldsymbol{h}$ with c's height replaced by h_c. Now, continuously lower h_c, and maintain the optimum surface by flips as below, until h_c attains the original value as given by $\boldsymbol{h}$ and the surface becomes $\mathcal{F}^*$.

If the diagonal d stays convex throughout, then no flips are performed. Otherwise, right after the surface becomes reflex at d, we flip d. This flip is either planarizing or convexifying. It does not change the convexity of any other edge, though, and thus restores optimality. To describe the generic step, let F_c be the part of the surface modified by flips so far. F_c is separated by diagonals $d_1, \ldots, d_k$ from its complement in the surface. Therefore, when decreasing h_c, all edges of the surface except possibly $d_1, \ldots, d_k$ stay convex. As soon as one of $d_1, \ldots, d_k$ becomes reflex, the next flip is performed. This restores optimality and extends F_c by one facet.

The number of flips is $O(n)$ because each flip extends F_c. It remains to be observed that exactly the same sequence of flips can be applied if the facet f is attached to $\mathcal{F}^*(P \setminus \Delta)$ with c's original height: Each diagonal d flipped above now becomes reflex as soon as d bounds the current locally convex part incident to c. Thus flipping d indeed constitutes an improving flip. ∎

We remark that each edge constructed above corresponds to an edge of the shortest-path tree $\pi_c(P)$ for P and c, defined in subsection 3.1.

LEMMA 7.5. *Let P be a simple polygon with n vertices, and let $\boldsymbol{h}$ be some height vector for $vert(P)$. The optimum $\mathcal{F}^*$ can be constructed by an improving flip sequence of length $O(n^2)$, from any given triangular surface for P and $\boldsymbol{h}$.*

Proof. We construct $\mathcal{F}^*$ by simulating an incremental algorithm. Let $\mathcal{T}$ be the triangulation to which the initial surface projects. Traverse the triangles of $\mathcal{T}$ in some adjacency order. Let Q be the union of the triangles visited so far. We maintain the following invariant for the current surface $\mathcal{F}$. The restriction of $\mathcal{F}$ to the polygon Q is optimum for Q (i.e., maximal and locally convex), and the restriction of $\mathcal{F}$ to $P \setminus Q$ coincides with the initial triangular surface.

The generic step processed a triangle Δ of $\mathcal{T}$ by constructing the optimum surface for the polygon $Q \cup \Delta$. As Δ is an ear of $Q \cup \Delta$, this takes $O(n)$ improving flips by Lemma 7.4. These flips leave the triangular surface for $P \setminus (Q \cup \Delta)$ unaffected, because all its vertices are complete. The number of steps (triangles) is $n - 3$. ∎

8. (Constrained) regular complexes. The concept of local convexity of polyhedral surfaces is closely related to the concept of constrained regularity of polygonal partitions. Let us elaborate on the consequences of the results in section 7 in terms of constrained regularity.

8.1. Regular triangulations. A triangulation $\mathcal{T}$ of a point set S is called *regular* if there exists some convex function on $conv(S)$ whose set of nonlinearity coincides with the set of internal edges of $\mathcal{T}$. Regular triangulations arise as duals of power diagrams [8] (a generalization of Voronoi diagrams) and have several applications [24, 23]. Figure 7.3 shows a well-known example of a nonregular triangulation.

We discuss the implications of section 7 for regular triangulations, that is, for the case $P = conv(S)$. For each height vector $\boldsymbol{h}$, the surface $\mathcal{F}^*$ is convex in this case and is the lower part, $low(S_{\boldsymbol{h}})$, of the boundary of the convex hull of the lifted point set $S_{\boldsymbol{h}}$. A unique regular triangulation $\mathcal{T}(S)_{\boldsymbol{h}}$ for S and $\boldsymbol{h}$ is obtained by projecting $low(S_{\boldsymbol{h}})$. Lemma 7.3 implies the following result.

THEOREM 8.1. *Let $\mathcal{T}$ and $\boldsymbol{h}$ be a given triangulation and a height vector, respectively, for a planar set S of n points. Then $\mathcal{T}$ can be flipped to the regular triangulation $\mathcal{T}(S)_{\boldsymbol{h}}$ using $O(n^2)$ flips, which are improving in the corresponding surfaces.*

Note that the *upper* part of the boundary of the convex hull of $S_{\boldsymbol{h}}$ can be obtained as well, by reflecting $S_{\boldsymbol{h}}$ at the plane $z = 0$ first. This yields another interesting result.

COROLLARY 8.2. *The convex hull of a three-dimensional n-point set M can be constructed by taking an arbitrary triangular surface for M and performing $O(n^2)$ improving surface flips.*

Observe that vertices of S may be absent in $\mathcal{T}(S)_{\boldsymbol{h}}$, namely, if some points of $S_{\boldsymbol{h}}$ lie strictly above $low(S_{\boldsymbol{h}})$. We call $\boldsymbol{h}$ a *convex* height vector (for S) if no point of $S_{\boldsymbol{h}}$ lies strictly above $low(S_{\boldsymbol{h}})$. If $\boldsymbol{h}$ is convex, then no (proper) pseudotriangle is ever created when flipping to regularity: Pseudotriangles stem from planarizing flips, which are never performed because the respective edges are convex already. Therefore all convexifying flips are tetrahedral flips, and all vertices show up in $\mathcal{T}(S)_{\boldsymbol{h}}$. In particular, if $S_{\boldsymbol{h}}$ happens to lie on a paraboloid with a vertical axis of rotation (say, $z = \boldsymbol{x}^2$), then $\mathcal{T}(S)_{\boldsymbol{h}}$ is the Delaunay triangulation $DT(S)$ of S; see, e.g., [26].

Remarks. It is well known that each triangulation of S can be transformed into $DT(S)$ by flips that are convexifying and tetrahedral in the respective surface (so-called Delaunay flips), but not every regular triangulation can be constructed in this way. Theorem 8.1 tells us that pseudotriangulations—rather than triangulations—are the right framework for flipping to regularity.

$O(n^2)$ convexifying flips always suffice for generating $DT(S)$. On the other hand, $\Omega(n^2)$ is a lower bound, even if S is in convex position; see, e.g., [26]. This implies that Theorem 8.1 is optimal in the worst case. Theorem 8.1 is a stronger version of a result in [25], which asserts that incremental insertion of points, and application of tetrahedral flips to the special triangular surface that arises from each insertion, is capable of generating the regular triangulation in $O(n^2)$ flips.

Recently, the quest for a *Delaunay pseudotriangulation* of a point set S has been formulated [47]. If "Delaunay" is interpreted as "convex at each edge of the projection surface," and if the *entire* class of pseudotriangulations for S is taken as the underlying setting (rather than some class of fixed edge rank), then our results imply that $\mathcal{T}(S)_{\boldsymbol{h}}$ (or $DT(S)$, respectively) is the solution. The Delaunay *minimum* pseudotriangulation does not exist in this setting unless S is in convex position: No minimum pseudotriangulation of S is regular if S contains some internal vertex v, because the incompleteness of v forces every surface to be nonconvex in the neighborhood of v.

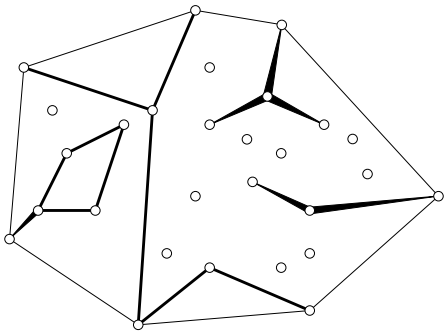


FIG. 8.1. *Cutting conv(S) along G.*

8.2. Constrained regular pseudotriangulations. Let us return to general augmented polygons (P, S). Here the maximal locally convex surface $\mathcal{F}^*$ is not convex continuable on the exterior of P in general. Thus $\mathcal{F}^*$ disagrees with the lower convex hull of the lifted point set $S_{\boldsymbol{h}}$ and contains pseudotriangular facets. The pseudotriangulation that $\mathcal{F}^*$ projects to always contains the edges of P. It can be interpreted as the regular pseudotriangulation for S *constrained* by P. Constrained triangulations, and especially, constrained *Delaunay* triangulations are well-known and versatile concepts; see, e.g., [40, 19, 4]. Very recently, constrained *regular* triangulations have been considered [13, 50]. This structure is not guaranteed to exist for general height vectors $\boldsymbol{h}$, however. We intend to show that our framework allows for a definition of constrained regular pseudotriangulations that always exist and are unique.

To exploit our concepts more generally, let G be a planar straight-line graph with vertex set S. Local convexity of a function f generalizes naturally, by requiring f to be convex on each line segment in $conv(S)$ that does not cross G. We call G *admissible* if, apart from the isolated vertices of G, each component of G is connected to the boundary of $conv(S)$. An admissible graph G defines a partition of $conv(S)$ as follows; see Figure 8.1. Cut $conv(S)$ along G, and move it apart at the cuts infinitesimally. This gives a collection of augmented polygons (Q_i, S_i) with $\bigcup_i S_i = S$.

Theorems 7.1 and 7.2 directly apply to this setting. Let $\mathcal{T}$ be a triangulation of S that includes all the edges of G. For a given height vector $\boldsymbol{h}$ for S, consider the unique triangular surface for $\mathcal{T}$ and $\boldsymbol{h}$. Let $\mathcal{F}_i$ denote the restriction of this surface to the polygon Q_i. Then $\mathcal{F}_i$ can be made locally convex by flipping away all reflex edges that do not belong to G. This gives a unique pseudotriangulation $\mathcal{PT}_i$ for each augmented polygon (Q_i, S_i). Altogether, a unique constrained regular pseudotriangulation $\mathcal{PT}(G)_{\boldsymbol{h}}$ for G and $\boldsymbol{h}$ is obtained.

The corresponding maximal locally convex function $f_{G,\boldsymbol{h}}$ on the domain $conv(S)$ is piecewise linear but, in general, not continuous. Its set of discontinuity is a subset of G and occurs at certain pseudotriangular faces of $\mathcal{PT}(G)_{\boldsymbol{h}}$, namely, where local convexity (with respect to G) and maximality (with respect to $\boldsymbol{h}$) contradict. The graph of $f_{G,\boldsymbol{h}}$ thus is no surface in our sense. We summarize as follows.

THEOREM 8.3. *Let S be a set of n points in the plane, and let G and $\boldsymbol{h}$, respectively, be an admissible straight-line graph and a height vector for S. There exists a unique constrained regular pseudotriangulation $\mathcal{PT}(G)_{\boldsymbol{h}}$ for G and $\boldsymbol{h}$. $\mathcal{PT}(G)_{\boldsymbol{h}}$ can be constructed by improving surface flips, starting from any triangulation of S that conforms with G. Moreover, the $O(n \log n)$ bound in Theorem 7.2 applies.*

Remarks. If G partitions $conv(S)$ into augmented *convex* polygons, then no incomplete vertices occur in $\mathcal{PT}(G)_{\boldsymbol{h}}$, which thus is a triangulation for every $\boldsymbol{h}$. The graph $F_{G,\boldsymbol{h}}$ of $f_{G,\boldsymbol{h}}$ then is a surface, namely, the continuous concatenation of lower convex hulls on top of these convex polygons.

There are other interesting instances of admissible constraining graphs G. For example, G may define a minimum pseudotriangulation $\mathcal{M}$ of S. In this case, $\mathcal{PT}(G)_{\boldsymbol{h}}$ gives ways of canonically refining $\mathcal{M}$. Alternatively, G may define a spanning tree of S, a choice used in [4] to transform crossing-free spanning trees into each other.

If the height vector $\boldsymbol{h}$ is convex, then all vertices of S show up in $\mathcal{PT}(G)_{\boldsymbol{h}}$. Moreover, only triangles are generated in the flipping process. $\mathcal{PT}(G)_{\boldsymbol{h}}$ is a triangulation, and $f_{G,\boldsymbol{h}}$ is continuous. The surface $F_{G,\boldsymbol{h}}$ is reflex only at edges of G, namely, at those that are absent in the regular triangulation $\mathcal{T}(S)_{\boldsymbol{h}}$. In the special case where the lifted point set $S_{\boldsymbol{h}}$ lies on the paraboloid $z = \boldsymbol{x}^2$, the constrained Delaunay triangulation [40] of S and G is obtained.

If the graph G is not admissible, then G splits $conv(S)$ into augmented polygons with possible holes. $\mathcal{PT}(G)_{\boldsymbol{h}}$ is still uniquely defined, because Theorem 5.1 generalizes to augmented polygons with holes in a straightforward way. However, the constrained regular complex $\mathcal{PT}(G)_{\boldsymbol{h}}$ fails to be a pseudotriangulation in general because pseudotriangular facets with holes may arise. It is worth mentioning that the polytope constructed in section 9 generalizes to representing arbitrary constrained regular complexes.

An attempt to define constrained regular complexes as full triangulations results in one of two phenomena: If convexity on all edges different from G is required, then the complex does not always exist; on the other hand, ambiguity occurs if reflex edges not belonging to G are admitted, even if they are all nonflippable (in the classical sense). Their structure depends on the flip sequence rather than on G and $\boldsymbol{h}$ alone. That is, several solutions (local optima) do exist in general. See Figure 7.2, where G delineates a nonconvex polygon.

9. Polytope representation. Let S be a set of n points in the plane. Let G denote an admissible straight-line graph on S, as defined in subsection 8.2. We establish the existence of a convex polytope that represents all the regular pseudotriangulations on S and constrained by G. This generalizes the polytope constructions in [39] (the associahedron) and in [28] (the secondary polytope, see also [15]), which concern the (unconstrained) regular triangulations of a point set S.

Let S' be a subset of S, which includes all the vertices of the (unique) connected component formed by G and the boundary of $conv(S)$. We consider the set $\Pi(G)$ of all pseudotriangulations $\mathcal{PT}$ that live on such subsets S' and conform with G. Clearly, for each height vector $\boldsymbol{h}$ for S, the constrained regular pseudotriangulation $\mathcal{PT}(G)_{\boldsymbol{h}}$ is a member of $\Pi(G)$.

We construct, for each $\mathcal{PT} \in \Pi(G)$ and each $\boldsymbol{h}$, a unique piecewise linear function $f_{\mathcal{PT},\boldsymbol{h}}$ on the domain $conv(S)$. Recall from subsection 8.2 that the admissible graph G defines a partition of $conv(S)$ into polygons Q_i. Let $\mathcal{PT}_i$ denote the part of $\mathcal{PT}$ inside Q_i. Then $\boldsymbol{h}$ assigns a height to each complete vertex of the pseudotriangulation $\mathcal{PT}_i$. Thus, by the surface theorem (Theorem 5.1), there exists a unique polyhedral surface $\mathcal{F}_i$ for $\mathcal{PT}_i$ and $\boldsymbol{h}$. The concatenation of the surfaces $\mathcal{F}_i$ is the graph of the function $f_{\mathcal{PT},\boldsymbol{h}}$ to be constructed. Note that $f_{\mathcal{PT},\boldsymbol{h}}$ is discontinuous in general.

Recall further that $f_{\mathcal{PT},\boldsymbol{h}}$ is locally convex with respect to G exactly if $\mathcal{PT} = \mathcal{PT}(G)_{\boldsymbol{h}}$. The optimality theorem (Theorem 7.1) thus implies $f_{\mathcal{PT},\boldsymbol{h}} \leq f_{\mathcal{PT}',\boldsymbol{h}}$ for all

$\mathcal{PT}' \in \Pi(G)$. Integrating both sides yields

$$\int_{x \in conv(S)} f_{\mathcal{PT},\boldsymbol{h}}(x)\, dx \leq \int_{x \in conv(S)} f_{\mathcal{PT}',\boldsymbol{h}}(x)\, dx. \tag{9.1}$$

Consider the left-hand-side integral in (9.1). The expressed volume is a linear (and homogenous) function of $\boldsymbol{h}$, because $f_{\mathcal{PT},\boldsymbol{h}}$ is piecewise linear and, by Lemma 5.2, the value of $f_{\mathcal{PT},\boldsymbol{h}}$ at each vertex of $\mathcal{PT}$ linearly depends on $\boldsymbol{h}$. Let $\boldsymbol{q}(\mathcal{PT})$ be the coefficient vector of this linear function. As analogous observations hold for the right-hand side in (9.1), we obtain

$$\boldsymbol{q}(\mathcal{PT}) \cdot \boldsymbol{h} \leq \boldsymbol{q}(\mathcal{PT}') \cdot \boldsymbol{h} \quad \text{for all } \mathcal{PT}' \in \Pi(G). \tag{9.2}$$

We now interpret $\boldsymbol{q}(\mathcal{PT})$ as a point in R^n and consider the convex polytope

$$\mathcal{Q}(G) = conv\{\boldsymbol{q}(\mathcal{PT}) \mid \mathcal{PT} \in \Pi(G)\}.$$

Further, we define the following set of height vectors:

$$\mathsf{V}(G, \mathcal{PT}) = \{\boldsymbol{h} \mid \mathcal{PT} = \mathcal{PT}(G)_{\boldsymbol{h}}\}.$$

Inequality (9.2) implies that, for each $\boldsymbol{h} \in \mathsf{V}(G, \mathcal{PT})$, the set $\{\boldsymbol{q}(\mathcal{PT}) \mid \mathcal{PT} \in \Pi(G)\}$ lies in a half-space whose boundary contains $\boldsymbol{q}(\mathcal{PT})$. Thus $\boldsymbol{q}(\mathcal{PT})$ is a vertex of the polytope $\mathcal{Q}(G)$ if and only if $\mathcal{PT} = \mathcal{PT}(G)_{\boldsymbol{h}}$ for some height vector $\boldsymbol{h}$. Note that $\boldsymbol{q}(\mathcal{PT})$ lies in the interior of $\mathcal{Q}(G)$ if $\mathcal{PT}$ is not constrained regular.

The collection of all sets $\mathsf{V}(G, \mathcal{PT})$, for $\mathcal{PT} \in \Pi(G)$, defines a cell complex $\mathcal{C}(G)$ in R^n because $\mathcal{PT}(G)_{\boldsymbol{h}}$ exists and is unique for each $\boldsymbol{h}$, by Theorem 8.3. $\mathsf{V}(G, \mathcal{PT})$ contains, for each member $\boldsymbol{h}$, the vector $\lambda \cdot \boldsymbol{h}$ for $\lambda \geq 0$. Moreover, $\boldsymbol{h}_1, \boldsymbol{h}_2 \in \mathsf{V}(G, \mathcal{PT})$ implies $\lambda \cdot \boldsymbol{h}_1 + (1-\lambda) \cdot \boldsymbol{h}_2 \in \mathsf{V}(G, \mathcal{PT})$ for $0 < \lambda < 1$. That is, $\mathsf{V}(G, \mathcal{PT})$ is a convex polyhedral cone with apex 0.

Consider a facet $g = \mathsf{V}(G, \mathcal{PT}_1) \cap \mathsf{V}(G, \mathcal{PT}_2)$ of the complex $\mathcal{C}(G)$. For $\boldsymbol{h} \in g$ we have $\boldsymbol{h} \cdot (q(\mathcal{PT}_1) - q(\mathcal{PT}_2)) = 0$ by inequality (9.2). Thus the line segment connecting $q(\mathcal{PT}_1)$ and $q(\mathcal{PT}_2)$ is normal to g. This implies that the edges of $\mathcal{Q}(G)$ are dual to the facets of $\mathcal{C}(G)$. For $\boldsymbol{h}$ lying in the relative interior of such a facet, there are exactly two constrained regular pseudotriangulations $\mathcal{PT}_1$ and $\mathcal{PT}_2$ that yield the same function $f_{\mathcal{PT}_1,\boldsymbol{h}} = f_{\mathcal{PT}_2,\boldsymbol{h}}$. Therefore, the graphs of $f_{\mathcal{PT}_1,\boldsymbol{h}_1}$ and $f_{\mathcal{PT}_2,\boldsymbol{h}_2}$, for $\boldsymbol{h}_1 \in \mathsf{V}(G, \mathcal{PT}_1)$ and $\boldsymbol{h}_2 \in \mathsf{V}(G, \mathcal{PT}_2)$, differ by a minimum combinatorial change, which corresponds to a flip that transforms $\mathcal{PT}_1$ and $\mathcal{PT}_2$ into each other. We conclude the following.

THEOREM 9.1. *Let S be a finite set of points in the plane, and let G be an admissible straight-line graph spanned by S. There exists a convex polytope $Q(G)$ for G, whose vertices are in one-to-one correspondence with the regular pseudotriangulations of S constrained by G. Moreover, the edges of this polytope correspond to flips as in section 2.*

Remarks. The existence of $\mathcal{Q}(G)$ implies that any two constrained regular pseudotriangulations $\mathcal{PT}(G)_{\boldsymbol{h}}, \mathcal{PT}(G)_{\boldsymbol{h}'} \in \Pi(G)$ are connected by a sequence of flips that retain constrained regularity. That is, there exists a sequence $\boldsymbol{h} = \boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_k = \boldsymbol{h}'$ of height vectors such that the functions $f_{G,\boldsymbol{h}_i}$ are all locally convex with respect to G. If G is chosen such that $f_{G,\boldsymbol{h}_i}$ is continuous (which, for instance, is the case if G defines a convex partition of $conv(S)$), then a tool for morphing locally convex surfaces is obtained. We raise the question for the maximal length k of the sequence above, that is, for the diameter of $\mathcal{Q}(G)$. The diameter obviously depends on both $n = |S|$ and G.

It is not hard to see that every pseudotriangulation of a simple polygon P (without internal vertices) is constrained regular. The corresponding polytope $\mathcal{Q}(P)$ therefore represents *all* possible pseudotriangulations of P. Moreover, any two pseudotriangulations of P can be transformed into each other with $O(n)$ flips; see subsection 3.1. We obtain the following result.

COROLLARY 9.2. *Let P be a simple polygon with n vertices. There exists a polytope $\mathcal{Q}(P)$ whose vertices and edges, respectively, represent all possible pseudotriangulations of P and the flips between them. The diameter of $\mathcal{Q}(P)$ is $O(n)$.*

Corollary 9.2 shows that, when generalizing from convex to nonconvex polygons, the diameter of the associahedron [39] remains linear.

Is there a polytope representation of all possible pseudotriangulations of an augmented polygon? One does exist for the subclass of all minimum pseudotriangulations of a point set S, by results in [47]. In view of our extended repertoire of flips, the general question arises naturally. We recently learned that an affirmative answer for the point set case has been given in [41], by generalizing the construction in [47]. Our results on flip distances in section 3 imply that the diameters of the polytopes in [47] and [41] are $O(n \log^2 n)$ and $O(n \log n)$, respectively. What is the relation of these polytopes to the polytopes in Theorem 9.1 and Corollary 9.2?

10. Conclusion. We have demonstrated that pseudotriangulations enjoy rich geometric and combinatorial properties, a new contribution to the theory of pseudotriangulations. Several key concepts for triangulations have been generalized to pseudotriangulations. A series of algorithmic implications have been described, which we believe to be of value in practical applications. The three main ingredients have been (1) the derivation of a new type of flip, (2) a proof of the realizability of pseudotriangulations as polyhedral surfaces, and (3) the establishment of a link to locally convex functions. For a better understanding of pseudotriangulations, several notions have been revitalized or introduced for the first time, like the edge rank, the completeness status of vertices, and the stability of pseudotriangulations. Let us conclude this paper with an extension of our results which seems most interesting to us.

The relationship between pseudotriangulations and locally convex functions, expressed in Theorem 7.2, opens a way to define pseudocomplexes in higher dimensions and flips therein. Also, the techniques used to prove the surface theorem (Theorem 5.1) do not depend on the underlying dimension. These tools may finally resolve the "3D pseudotriangulation mystery." We are exploring the consequences of this approach in a separate paper.

Acknowledgments. The authors gratefully acknowledge insightful discussions on the presented topic with Günter Rote and Francisco Santos.

REFERENCES

[1] P. K. AGARWAL, J. BASCH, L. J. GUIBAS, J. HERSHBERGER, AND L. ZHANG, *Deformable free space tilings for kinetic collision detection*, in Algorithmic and Computational Robotics: New Directions (Proceedings of the 5th Workshop on the Algorithmic Foundations of Robotics), B. R. Donald, K. Lynch, and D. Rus, eds., 2001, pp. 83–96.

[2] H. AHN, S.-W. CHENG, O. CHEONG, AND J. SNOEYINK, *The reflex-free hull*, in Proceedings of the 13th Canadian Conference on Computational Geometry, Waterloo, Canada, 2001, pp. 9–12.

[3] O. AICHHOLZER AND F. AURENHAMMER, *Straight skeletons for general polygonal figures in the plane*, in Voronoi's Impact on Modern Sciences II, A. M. Samoilenko, ed., Proc. Inst. Math. Nat. Acad. Sci. Ukraine 21, Kiev, Ukraine, 1998, pp. 7–21.

[4] O. Aichholzer, F. Aurenhammer, and F. Hurtado, *Sequences of spanning trees and a fixed tree theorem*, Comput. Geom., 21 (2002), pp. 3–20.
[5] O. Aichholzer, F. Aurenhammer, H. Krasser, and B. Speckmann, *Convexity minimizes pseudo-triangulations*, in Proceedings of the 14th Canadian Conference on Computational Geometry, Lathbridge, Alberta, Canada, 2002, pp. 158–161.
[6] O. Aichholzer, M. Hoffmann, B. Speckmann, and C. D. Tóth, *Degree Bounds for Constrained Pseudo-Triangulations*, in Proceedings of the 15th Canadian Conference on Computational Geometry, Halifax, Canada, 2003, pp. 155–158.
[7] M. Ajtai, V. Chvátal, M. M. Newborn, and E. Szemerédi, *Crossing-free subgraphs*, Ann. Discrete Math., 12 (1982), pp. 9–12.
[8] F. Aurenhammer, *Power diagrams: Properties, algorithms, and applications*, SIAM J. Comput., 16 (1987), pp. 78–96.
[9] F. Aurenhammer, *A criterion for the affine equivalence of cell complexes in R^d and convex polyhedra in R^{d+1}*, Discrete Comput. Geom., 2 (1987), pp. 49–64.
[10] F. Aurenhammer, *A relationship between Gale transforms and Voronoi diagrams*, Discrete Appl. Math., 28 (1990), pp. 83–91.
[11] F. Aurenhammer and Y.-F. Xu, *Optimal triangulations*, in Encyclopedia of Optimization, Vol. 4, C. A. Floudas and P. M. Pardalos, eds., Kluwer Academic Publishing, Dordrecht, The Netherlands, 2000, pp. 160–166.
[12] S. Bespamyatnikh, *Transforming Pseudo-Triangulations*, manuscript, Department of Computer Science, University of Texas at Dallas, Dallas, TX, 2003.
[13] S. Balaven, C. Bennis, J.-D. Boissonnat, and M. Yvinec, *Conforming orthogonal meshes*, in Proceedings of the 11th International Meshing Roundtable, Ithaca, NY, 2002, pp. 219–230.
[14] M. Bern and D. Eppstein, *Mesh generation and optimal triangulation*, in Computing in Euclidean Geometry, D.-Z. Du and F. Hwang, eds., Lecture Notes Ser. Comput. 4, World Scientific, River Edge, NJ, 1995, pp. 47–123.
[15] L. J. Billera, P. Filliman, and B. Sturmfels, *Constructions and complexity of secondary polytopes*, Adv. Math., 83 (1990), pp. 155–179.
[16] H. Brönnimann, L. Kettner, M. Pocchiola, and J. Snoeyink, *Counting and Enumerating Pseudo-Triangulations with the Greedy Flip Algorithm*, manuscript, Polytechnic University, Brooklyn, NY, 2001.
[17] B. Chazelle, *A theorem on polygon cutting with applications*, in Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (FOCS), Chicago, IL, 1982, pp. 339–349.
[18] B. Chazelle, H. Edelsbrunner, M. Grigni, L. J. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink, *Ray shooting in polygons using geodesic triangulations*, Algorithmica, 12 (1994), pp. 54–68.
[19] L. P. Chew, *Constrained Delaunay triangulations*, Algorithmica, 4 (1989), pp. 97–108.
[20] H. Crapo and W. Whiteley, *Spaces of stresses, projections, and parallel drawings of spherical polyhedra*, Beiträge Algebra Geom., 35 (1994), pp. 259–281.
[21] L. De Floriani, B. Falcidieno, C. Pienovi, and G. Nagy, *On Sorting Triangles in a Delaunay Tesselation*, Technical report, Inst. Mat. Appl., Consiglio Nazionale della Richerche, Genova, Italy, 1988.
[22] H. Edelsbrunner, *An acyclicity theorem for cell complexes in d dimensions*, Combinatorica, 10 (1990), pp. 251–260.
[23] H. Edelsbrunner, *The union of balls and its dual shape*, Discrete Comput. Geom., 13 (1995), pp. 415–440.
[24] H. Edelsbrunner and E. P. Mücke, *Three-dimensional alpha shapes*, ACM Trans. Graph., 13 (1994), pp. 43–72.
[25] H. Edelsbrunner and N. R. Shah, *Incremental topological flipping works for regular triangulations*, Algorithmica, 15 (1996), pp. 223–241.
[26] S. Fortune, *Voronoi diagrams and Delaunay triangulations*, in Computing in Euclidean Geometry, D.-Z. Du and F. Hwang, eds., Lecture Notes Ser. Comput. 4, World Scientific, River Edge, NJ, 1995, pp. 225–265.
[27] J. Friedman, J. Hershberger, and J. Snoeyink, *Efficiently planning compliant motion in the plane*, SIAM J. Comput., 25 (1996), pp. 562–599.
[28] I. M. Gel'fand, M. M. Kapranov, and A. V. Zelevinsky, *Newton polyhedra of principal A-determinants*, Soviet Math. Dokl., 308 (1989), pp. 20–23.
[29] M. T. Goodrich and R. Tamassia, *Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations*, J. Algorithms, 23 (1997), pp. 51–73.
[30] R. Haas, D. Orden, G. Rote, F. Santos, B. Servatius, H. Servatius, D. Souvaine, I. Streinu, and W. Whiteley, *Planar minimally rigid graphs and pseudo-triangulations*,

Comput. Geom., (2003), pp. 154–163.
[31] S. Hanke, T. Ottmann, and S. Schuierer, *The edge-flipping distance of triangulations*, J. Univ. Comput. Sci., 2 (1996), pp. 570–579.
[32] J. Hershberger, *An optimal visibility graph algorithm for triangulated simple polygons*, Algorithmica, 4 (1989), pp. 141–155.
[33] C. Huemer, *Flip Operations for Combinatorial and Geometric Objects*, Masters thesis, Institute for Theoretical Computer Science, Graz University of Technology, Graz, Austria, 2003.
[34] F. Hurtado, M. Noy, and J. Urrutia, *Flipping edges in triangulations*, Discrete Comput. Geom., 22 (1999), pp. 333–346.
[35] L. Kettner, D. Kirkpatrick, A. Mantler, J. Snoeyink, B. Speckmann, and F. Takeuchi, *Tight degree bounds for pseudo-triangulations of points*, Comput. Geom., 25 (2003), pp. 3–12.
[36] D. Kirkpatrick, J. Snoeyink, and B. Speckmann, *Kinetic collision detection for simple polygons*, Internat. J. Comput. Geom. Appl., 12 (2002), pp. 3–27.
[37] D. Kirkpatrick and B. Speckmann, *Kinetic maintenance of context-sensitive hierarchical representations for disjoint simple polygons*, in Proceedings of the 18th Annual ACM Symposium on Computational Geometry, Barcelona, Spain, 2002, pp. 179–188.
[38] C. L. Lawson, *Properties of n-dimensional triangulations*, Comput. Aided Geom. Design, 3 (1986), pp. 231–246.
[39] C. W. Lee, *The associahedron and triangulations of the n-gon*, European J. Combin., 10 (1989), pp. 173–181.
[40] D. T. Lee and A. K. Lin, *Generalized Delaunay triangulation for planar graphs*, Discrete Comput. Geom., 1 (1986), pp. 201–217.
[41] D. Orden and F. Santos, *The Polyhedron of Non-crossing Graphs on a Planar Point Set*, manuscript, Universidad de Cantabria, Santander, Spain, 2002.
[42] M. Pocchiola and G. Vegter, *Minimal tangent visibility graphs*, Comput. Geom., 6 (1996), pp. 303–314.
[43] M. Pocchiola and G. Vegter, *Topologically sweeping visibility complexes via pseudo-triangulations*, Discrete Comput. Geom., 16 (1996), pp. 419–453.
[44] M. Pocchiola and G. Vegter, *On polygon covers*, in Advances in Discrete and Computational Geometry, B. Chezeck, D. E. Goodman, and R. Pollack, eds., Contemp. Math. 223, AMS, Providence, RI, 1999, pp. 257–258.
[45] D. Randall, G. Rote, F. Santos, and J. Snoeyink, *Counting triangulations and pseudo-triangulations of wheels*, in Proceedings of the 13th Canadian Conference on Computational Geometry, Waterloo, Canada, 2001, pp. 117–120.
[46] V. T. Rajan, *Optimality of the Delaunay triangulation in R^d*, Discrete Comput. Geom., 12 (1994), pp. 189–202.
[47] G. Rote, F. Santos, and I. Streinu, *Expansive motions and the polytope of pointed pseudo-triangulations*, in Discrete and Computational Geometry—The Goodman–Pollack Festschrift, B. Aronov, S. Basu, J. Pach, and M. Sharir, eds., Algorithms and Combinatorics, Springer, Berlin, 2003, pp. 699–736.
[48] G. Rote, C. A. Wang, L. Wang, and Y. Xu, *On Constrained Minimum Pseudo-Triangulations*, manuscript, Institut für Informatik, FU-Berlin, 2002.
[49] B. Speckmann and C. D. Tóth, *Allocating vertex π-guards in simple polygons via pseudo-triangulations*, in Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, SIAM, Philadelphia, 2003, pp. 109–118.
[50] J. R. Shewchuk, *Updating and constructing constrained Delaunay and constrained regular triangulations by flips*, in Proceedings of the 19th Annual ACM Symposium on Computational Geometry, 2003, pp. 181–190.
[51] I. Streinu, *A combinatorial approach to planar non-colliding robot arm motion planning*, in Proceedings of the 41st IEEE Symposium on Foundations of Computer Science (FOCS), Redondo Beach, CA, 2000, pp. 443–453.

SIAM J. COMPUT.
Vol. 32, No. 6, pp. 1654–1673

A SUBQUADRATIC SEQUENCE ALIGNMENT ALGORITHM FOR UNRESTRICTED SCORING MATRICES*

MAXIME CROCHEMORE†, GAD M. LANDAU‡, AND MICHAL ZIV-UKELSON§

Abstract. Given two strings of size n over a constant alphabet, the classical algorithm for computing the similarity between two sequences [D. Sankoff and J. B. Kruskal, eds., *Time Warps, String Edits, and Macromolecules*; Addison–Wesley, Reading, MA, 1983; T. F. Smith and M. S. Waterman, *J. Molec. Biol.*, 147 (1981), pp. 195–197] uses a dynamic programming matrix and compares the two strings in $O(n^2)$ time. We address the challenge of computing the similarity of two strings in subquadratic time for metrics which use a scoring matrix of unrestricted weights. Our algorithm applies to both *local* and *global* similarity computations. The speed-up is achieved by dividing the dynamic programming matrix into variable sized blocks, as induced by Lempel–Ziv parsing of both strings, and utilizing the inherent periodic nature of both strings. This leads to an $O(n^2/\log n)$, algorithm for an input of constant alphabet size. For most texts, the time complexity is actually $O(hn^2/\log n)$, where $h \leq 1$ is the entropy of the text. We also present an algorithm for comparing two *run-length* encoded strings of length m and n, compressed into m' and n' runs, respectively, in $O(m'n + n'm)$ complexity. This result extends to all distance or similarity scoring schemes that use an additive gap penalty.

Key words. alignment, dynamic programming, text compression, run length

AMS subject classifications. 74K05, 90C39, 68P30

DOI. 10.1137/S0097539702402007

1. Introduction.

Rapid progress in large-scale DNA sequencing has opened a new level of computational challenges in storing, organizing, and analyzing the wealth of resulting biological information. One of the most interesting new fields created by the availability of complete genomes is that of genome comparison. (The *genome* is all of the DNA sequence passed from one generation to the next.) Comparing complete genomes can give deep insights about the relationship between organisms, as well as shedding light on the function of specific genes in each single genome. The challenge of comparing complete genomes necessitates the creation of additional, more efficient computational tools.

One of the most common problems in biological comparative analysis is that of aligning two long bio-sequences in order to measure their similarity. The alignment is classically based on the transformation of one sequence into the other via operations

*Received by the editors February 3, 2002; accepted for publication (in revised form) July 24, 2003; published electronically October 14, 2003. This paper is the extended journal version of [11].
http://www.siam.org/journals/sicomp/32-6/40200.html

†Institut Gaspard-Monge, Université de Marne-la-Vallée, Cité Descartes, Champs-sur-Marne, 77454 Marne-la-Vallée Cedex 2, France, and Department of Computer Science, King's College of London, Strand, London WC2R 2LS, UK (mac@monge.univ-mlv.fr; http://www-igm.univ-mlv.fr/~mac/). This author's research was partially supported by the CNRS, NATO Science Programme, and the Wellcome Trust Foundation.

‡Department of Computer Science, Haifa University, Haifa 31905, Israel, and Department of Computer and Information Science, Polytechnic University, Six MetroTech Center, Brooklyn, NY 11201-3840 (landau@poly.edu). This author's research was partially supported by NSF grant CCR-0104307, by NATO Science Programme grant PST.CLG.977017, by Israel Science Foundation grants 173/98 and 282/01, by the FIRST Foundation of the Israel Academy of Science and Humanities, and by an IBM Faculty Partnership Award.

§Department of Computer Science, Haifa University, Haifa 31905, Israel (michal@cs.haifa.ac.il). This author is on education leave from the IBM T.J.Watson Research Center. This author's research was partially supported by Israel Science Foundation grants 173/98 and 282/01 and by the FIRST Foundation of the Israel Academy of Science and Humanities.

of substitutions, insertions, and deletions (indels). The costs of these transformations are given by a scoring matrix.

DEFINITION 1.1 (see Gusfield [25]). The global alignment problem. *Given a pairwise scoring matrix δ over the alphabet Σ, the similarity of two strings A and B is defined as the value* maxV *of the alignment of A and B that maximizes the total alignment value.*

- The score value $maxV$ is called the *optimal global alignment value* of A and B.
- A description of a $maxV$-scoring transformation of A into B is called a *global alignment trace*.

In many applications, two strings may not be highly similar in their entirety but may contain regions that are highly similar. The task is to find and extract a pair of regions, one from each of the two given strings, that exhibit high similarity. This is called the *local alignment* or *local similarity* and is defined formally below.

DEFINITION 1.2 (see Gusfield [25]). The local alignment problem. *Given two strings A and B, find substrings α and β of A and B, respectively, whose similarity (optimal global alignment value) is maximum over all pairs of substrings from A and B.*

- The score value $maxL$ of the most similar pair of substrings α and β is called the *optimal local alignment value*.
- The description of a $maxL$-scoring transformation of substring α into substring β is called a *local alignment trace*.

Given two strings of size n, both global and local similarity problems can be solved in $O(n^2)$ time by dynamic programming [25], [36], [50]. After the optimal similarity scores have been computed, both global alignment and local alignment traces can be reported in time linear with their size [10], [26], [28].

1.1. Results. In this paper data compression techniques are employed to speed up the alignment of two strings. The compression mechanism enables the algorithm to adapt to the data and to utilize its repetitions. The periodic nature of the sequence is quantified via its *entropy*, denoted by the real number h, $0 < h \leq 1$. Entropy is a measure of how "compressible" a sequence is (see [7], [13]), and is small when there is a lot of order (i.e, the sequence is repetitive and therefore more compressible) and large when there is a lot of disorder (see section 2.2).

Our results include the following algorithms.

1.1.1. Global alignment.

- We present an $O(n^2/\log n)$ algorithm for computing the optimal global alignment value of two strings, each of size n, over a constant alphabet (see section 3). The algorithm is even faster when the sequence is compressible. In fact, for most texts, the complexity of our algorithm is actually $O(hn^2/\log n)$.
- After the optimal score is computed, a single alignment trace corresponding to the optimal score can be recovered in time complexity that is linear with the size of the trace (see section 4).
- For global alignment over "discrete" scoring matrices, we explain how the space complexity can be reduced to $O(h^2n^2/(\log n)^2)$ without impairing the $O(hn^2/\log n)$ time complexity (see section 5).

1.1.2. Local alignment.

- We describe a subquadratic $O(hn^2/\log n)$ algorithm for the computation of the optimal local alignment value of two strings over a constant alphabet (see

section 6.1).
- Given an index on A where substring α ends and an index on B where substring β ends, an *optimal local alignment trace* can be reported in time linear with its size (see section 6.2).

1.1.3. Comparing two run-length encoded strings.
- We give an algorithm for comparing two *run-length* encoded strings of length m and n, compressed to m' and n' runs, respectively, using any distance or similarity scoring scheme with additive gaps, in $O(m'n + n'm)$ complexity (see section 7).

The algorithms described in this paper are the first to approach *fully LZ compressed* string alignment (both source and target strings are compressed). The methods given in this paper can also be used by applications where both input strings are stored or transmitted in the form of an $LZ78$ or LZW compressed sequence, thus providing an efficient solution to the problem of how to compare two strings without having to decompress them first.

Remark. For the sake of simplicity we assume, throughout the description of the global alignment and the local alignment solutions, that both input strings A and B are of the same size n, and that both sequences share the same entropy h. For the case of comparing string A of size m and entropy $0 < h_A \leq 1$ with string B of size n and entropy $0 < h_B \leq 1$, the results of subsections 1.1.1 and 1.1.2 are as follows:
- $O(mn(h_A/\log m + h_B/\log n))$ time and space complexity for both global alignment and local alignment replaces the $O(hn^2/\log n)$ result.
- $O(h_A h_B mn/\log m \log n)$ space complexity for global alignment over "discrete" scoring matrices replaces the $O(h^2n^2/(\log n)^2)$ result.

1.2. Previous results. The only previously known subquadratic global alignment string comparison algorithm, by Masek and Paterson [40], is based on the "Four Russians" paradigm. The Four Russians algorithm divides the dynamic programming table into uniform-sized ($\log n$ by $\log n$) blocks and uses table lookup to obtain an $O(n^2/\log n)$ time complexity string comparison algorithm, based on two assumptions. One assumption is that the sequence elements come from a constant alphabet. The other, which they denote the "discreteness" condition, is that the weights (of substitutions and indels) are all rational numbers.

Our algorithms present a new approach and are better than the above algorithm in two respects. First, the algorithms presented here are faster for compressible sequences. For such sequences, the complexity of our algorithms is $O(hn^2/\log n)$, where $h \leq 1$ is the entropy of the sequence.

Second, our algorithms are general enough to support scoring schemes with real number weights. For many scoring schemes, the rational number weights supported by Masek and Paterson's algorithm do not suffice. For example, the entries of PAM similarity matrices [25], as well as BLOSUM evolutionary distance matrices [25], are defined to be real numbers, computed as log-odds ratios, and therefore could be irrational.

The paper by Masek and Paterson concludes with the following statement: "The most important problem remaining is finding a better algorithm for the finite (in our terms constant) alphabet case without the discreteness condition." Here, more than twenty years later, this important open question will finally be answered!

The advantages of our approach are based on the following facts. First, our algorithm does not require any precomputation of lookup tables and therefore can afford more flexible weight values. Also, instead of dividing the dynamic programming

matrix into uniform-sized blocks as did Masek and Paterson, we employ a variable-sized block partition, as induced by Lempel–Ziv factorization of both source and target. The common denominator between blocks, maximized by the compression technique, is then recycled and used for computing the relevant information for each block, in time which is linear with the length of its sides. In this sense, the approach described in this paper can be viewed as another example of speeding up dynamic programming by keeping and computing only a relevant subset of important values, as demonstrated in [17], [18], [34], and [48]. A similar unbalanced strategy has been successfully used for square detection in strings [12] to speed up the original algorithm based on a divide-and-conquer approach [37].

2. Preliminaries.

2.1. The alignment graph. The dynamic programming solution to the string comparison computation problem can be represented in terms of a weighted alignment graph [25]; see Figure 2.1.

The weight of a given edge can be specified directly on the grid graph or, as is frequently the case in biological applications, is given by a scoring matrix, denoted δ, which specifies the substitution score for each pair of characters and the insertion/deletion scores for each character from the alphabet.

The two widely used classes of scoring schemes are distance scoring, in which the objective is to minimize the total alignment score, and similarity scoring, in which the objective is to maximize the total alignment score. Within these classes, scoring schemes are further characterized by the treatment of gap scores. A *gap* is the result of the deletion of one or more consecutive characters in one of the sequences. *Additive* gap scores assign a constant weight to each of the consecutive characters. For other gap functions which have been found useful for biological sequences, see [25]. The solutions in this paper assume a scoring scheme with additive gap scores.

Global alignment via dynamic programming. The classical dynamic programming algorithm for the global comparison of two strings will set the value at each vertex (i, j) of the alignment graph, row by row in a left to right order, to the score between the first i characters of A and the first j characters of B, using the following recurrence:

$$\begin{aligned} V(i,j) = max[&V(i,j-1) + \delta(\epsilon, B_j),\\ &V(i-1,j) + \delta(A_i, \epsilon),\\ &V(i-1,j-1) + \delta(A_i, B_j)]. \end{aligned}$$

Computing and setting the values of all vertices in the alignment graph, using the above recurrence, takes $O(n^2)$ time and space. After the values at each vertex of the alignment graph have been computed and set, the optimal global alignment value $maxV$ is found at vertex (n, n) of the graph.

If each vertex in the alignment graph stores the operation (insertion, deletion, substitution) selected when its value was set, then a global alignment trace, corresponding to an optimal path in the alignment graph, can be recovered in time linear with its size, starting from vertex (n, n), which contains the maximal score, and tracing the edges back up to vertex $(0, 0)$ in the graph.

Local alignment via dynamic programming. Smith and Waterman [50] (see also [25]) showed that essentially the same $O(|A||B|)$ dynamic programming solution can be used for computing local similarity, provided that the score of the alignment of two empty strings is defined as 0, and only pairs whose alignment scores are above 0 are of interest. The Smith–Waterman algorithm for computing local similarity computes

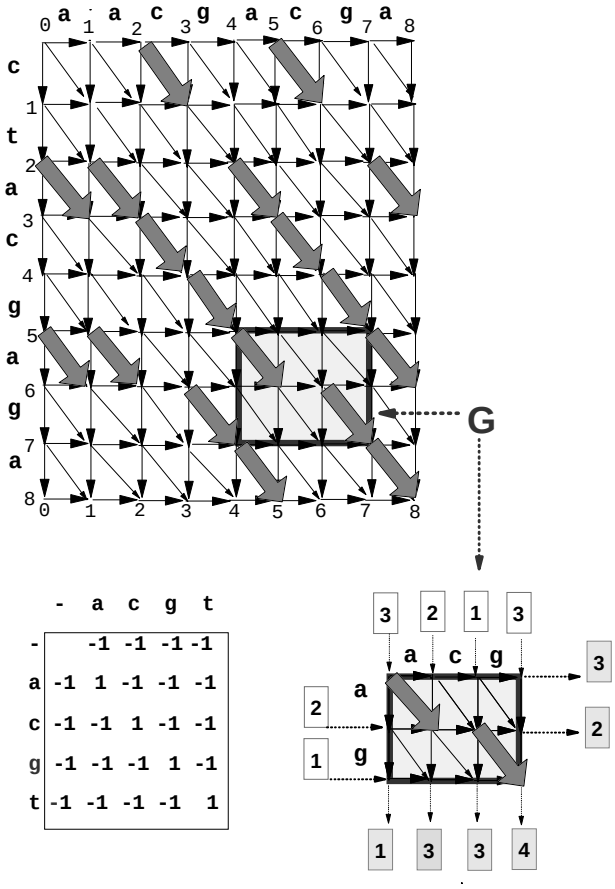


FIG. 2.1. *The alignment graph for comparing strings* A = "*ctacgaga*" *and* B = "*aacgacga*." *The scoring scheme matrix* δ *is shown in the lower left corner of the figure. The highest scoring global alignment paths originate in vertex* (0,0)*, end in vertex* (8,8)*, and have a total weight of* 3*. The highest scoring local alignment path has a total weight of* 5 *and corresponds to the alignment of substrings* a = *"acgaga" and* b = *"acgacga." A subgraph* G *corresponding to the block for comparing substrings* a = "*ag*" *and* b = "*acg*" *is shown in the lower-right corner of the figure. Also specified are the values* I *for the entries of the input border for* G *(in white-shaded rectangles), and the values* O *of the output border of* G *(in grey-shaded rectangles), as set during a local alignment computation.*

the following recurrence, which includes 0 as an additional option, and thus restricts the scores to nonnegative values:

$$\begin{aligned} L(i,j) = max[0, & L(i,j-1) + \delta(\epsilon, B_j), \\ & L(i-1,j) + \delta(A_i, \epsilon), \\ & L(i-1,j-1) + \delta(A_i, B_j)]. \end{aligned}$$

The method for computing the optimal local alignment value $maxL$ is to compute all alignment graph vertex values $L(i,j)$ in $O(n^2)$ time and space, and then find the largest value at *any* vertex on the table, say at vertex (i_{end}, j_{end}).

Given the vertex (i_{end}, j_{end}), which carries the score $maxL$, the corresponding substrings α and β giving the optimal local alignment of A and B are obtained in time

linear with their size, by using the stored operations (insertion, deletion, substitution) to trace back the edges from vertex (i_{end}, j_{end}) until a vertex (i_{start}, j_{start}) is reached that has value zero. Then the optimal local alignment substrings for vertex (i_{end}, j_{end}) are $\alpha = A[i_{start} \dots i_{end}]$ and $\beta = B[j_{start} \dots j_{end}]$ (see [25]).

2.2. A block partition of the alignment graph based on LZ78 factorization. The traditional aim of text compression is the efficient use of resources such as storage and bandwidth. Here, we will compress the sequences in order to speed up the alignment process. Note that this approach, denoted "acceleration by text compression," has been recently applied to a related problem—that of *exact string matching* [31], [39], [49].

It should also be mentioned that another related problem, that of exact string matching in compressed text without decoding it, which is often referred to as "compressed pattern matching," has been studied extensively [4], [19], [45]. Along these lines, string search in compressed text was developed for the compression paradigm of LZ78 [54] and its subsequent variant LZW [52], as described in [32], [46]. A more challenging problem is that of "fully compressed" pattern matching, when both the pattern and text strings are compressed [22], [23].

For the LZ78-LZW paradigm, compressed matching has been extended and generalized to *approximate pattern matching* (finding all occurrences of a short sequence within a long one, allowing up to k changes) in [30], [44].

The LZ compression methods are based on the idea of self–reference: while the text file is scanned, substrings or phrases are identified and stored in a dictionary, and whenever, later in the process, a phrase or concatenation of phrases is encountered again, this is compactly encoded by suitable pointers [35], [53], [54].

Of the several existing versions of the method, we will use those called the $LZ78$ family [52], [54]. The main feature which distinguishes $LZ78$ factorization from previous LZ compression algorithms is the choice of codewords. Instead of allowing pointers to reference any string that has appeared previously, the text seen so far is parsed into phrases, where each phrase is the longest matching phrase seen previously plus one character. For example, the string "S = aacgacg" is divided into fours phrases: a, ac, g, acg. Each phrase is encoded as an index to its prefix, plus the extra character. The new phrase is then added to the list of phrases that may be referenced.

Since each phrase is distinct from others, the following upper bound applies to the possible number of phrases obtained by $LZ78$ factorization.

THEOREM 2.1 (see Lempel and Ziv [35]). *Given a sequence S of size n over a constant alphabet, the maximal number of distinct phrases in S is $O(\frac{n}{\log n})$.*

Even though the upper bound above applies to any possible sequence over a constant alphabet, it has been shown that in many cases we can do better than that.

Intuitively, the $LZ78$ algorithm compresses the sequence because it is able to discover some repeated patterns. Therefore, in order to compute a tighter upper bound on the number of phrases obtained by $LZ78$ factorization for "compressible" sequences, the repetitive nature of the sequence needs to be quantified. One of the fundamental ideas in information theory is that of *entropy*, denoted by the real number h, $0 < h \leq 1$, which measures the amount of disorder or randomness, or inversely, the amount of order or redundancy in a sequence. Entropy is small when there is a lot of order (i.e, the sequence is repetitive) and large when there is a lot of disorder. The entropy of a sequence should ideally reflect the ratio between the size of the sequence after it has been compressed and the length of the uncompressed sequence.

The number of distinct phrases obtained by $LZ78$ factorization has been shown

to be $O(hn/\log n)$ for most texts [7], [13], [35], [51], [54]. Note that, for any text over a constant alphabet, the upper bound above still applies by setting h to 1.

3. Computing the optimal global similarity value.

3.1. Definitions and basic observations. The alignment graph will be partitioned as follows. Strings A and B will be parsed using $LZ78$ factorization. This induces a partition of the alignment graph, for comparing A with B, into variable-sized blocks (see Figure 3.1). Each block will correspond to a comparison of an LZ phrase of A with an LZ phrase of B.

Let xa denote a phrase in A obtained by extending a previous phrase x of A with character a, and yb denote a phrase in B obtained by extending a previous phrase of B with character b.

From now on we will focus on the computations necessary for a single block of the alignment graph.

Consider the block G which corresponds to the comparison of xa and yb. We define *input border* I as the left and top borders of G, and *output border* O as the bottom and right borders of G. (The node entries on the input border are numbered in a clockwise direction, and the node entries on the output border are numbered in a counterclockwise direction.)

Rather than filling in the values of each vertex in G, as does the classical dynamic programming algorithm, the only values computed for each block will be those on its I/O borders (see Figures 2.1 and 6.1A). Intuitively, this is the reason behind the efficiency gain.

Let ℓ_r denote the number of rows in G, $\ell_r = |xa|$. Let ℓ_c denote the number of

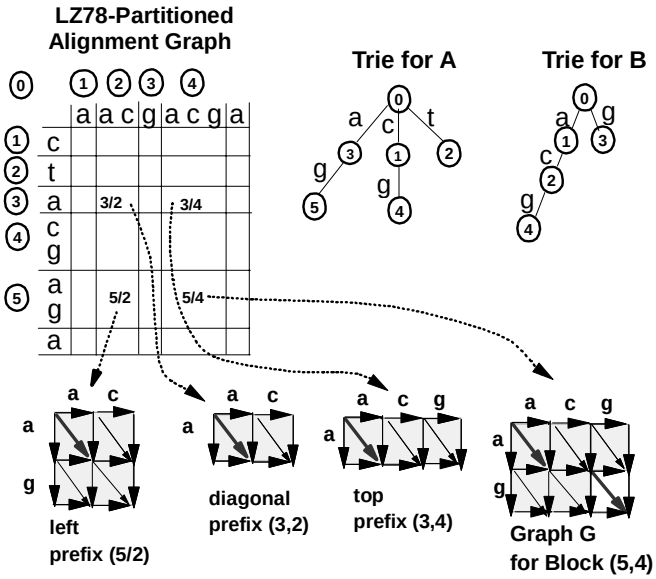


FIG. 3.1. *The block partition of the alignment graph, and the tries corresponding to* LZ78 *parsing of strings* A = "*ctacgaga*" *and* B = "*aacgacga*." *Note that for the block G in this example,* $\alpha =$ "*ag*," $\beta =$ "*acg*," $\ell_r = 2$, $\ell_c = 3$, $i = 5$, *and* $j = 4$. *(The new cell of* G, *which does not appear in any of the prefix blocks, is the rightmost cell at the bottom row of* G *and can be distinguished by its white color.) This figure continues Figure* 2.1.

columns in G, $\ell_c = |yb|$. Let $t = \ell_r + \ell_c$. Clearly, $|I| = |O| = t$.

We define the following three *prefix* blocks of G:

1. The *left prefix* of G denotes the block comparing phrase xa of A and phrase y of B.
2. The *diagonal prefix* of G denotes the block comparing phrase x of A and phrase y of B.
3. The *top prefix* of G denotes the block comparing phrase x of A and phrase yb of B.

Observation 1. When traversing the blocks of an $LZ78$ parsed alignment graph in a left-to-right, top-to-bottom order, the blocks for the left prefix, diagonal prefix, and top prefix of G are encountered prior to block G.

Note that the graph for the left prefix of G is identical to the subgraph of G containing all columns but the last one. More specifically, both the structure and the weights of edges of these two graphs are identical, but the weights to be assigned to vertices during the similarity computation may vary according to the input border values. Similarly, the graph for the top prefix block is identical in structure to a subgraph of G containing all rows but the last one, and the graph for the diagonal prefix block is similar in structure to the last subgraph of G which is obtained by removing both the last column and the last row of G. The only new cell in G, which does not appear in any of its prefix block graphs, is the cell for comparing a and b. This new cell consists of one new vertex and three new edges.

3.2. I/O propagation across G. The work for each block consists of two stages (a similar approach is shown in [8], [29], [34]):

1. *Encoding*: study the structure of G and represent it in an efficient way.
2. *Propagation*: given I and the encoding of G, constructed in the previous stage, compute O for G.

The structure of G is encoded by computing weights of optimal paths connecting each entry of its input border with each entry of its output border. The following $DIST$ matrix is used (see Figure 3.2).

DEFINITION 3.1. *$DIST[i, j]$ stores the weight of the optimal path from entry i of the input border of G to entry j of its output border.*

$DIST$ matrices have also been used in [5], [8], [29], [34], and [48].

Given input row I and the $DIST$ for G, the weight of output row vertex O_j can be computed as the maximum among the sums $I_r + DIST[r, j]$ if there is indeed a path connecting input border entry r with output border entry j.

Vertex O_j is the maximum of column j of the following OUT matrix, which merges the information from input row I and $DIST$. (See Figure 3.2.)

DEFINITION 3.2. $OUT[i, j] = I_i + DIST[i, j]$.

Aggarwal and Park [3] and Schmidt [48] observed that $DIST$ matrices are Monge arrays [43].

DEFINITION 3.3. *A matrix $M[0 \dots m, 0 \dots n]$ is* Monge *if either condition* 1 or 2 *below holds for all $a, b = 0 \dots m; c, d = 0 \dots n$:*

1. convex condition: $M[a, c] + M[b, d] \leq M[b, c] + M[a, d]$ *for all $a < b$ and $c < d$;*
2. concave condition: $M[a, c] + M[b, d] \geq M[b, c] + M[a, d]$ *for all $a < b$ and $c < d$.*

Since $DIST$ is Monge, so is OUT, which is a $DIST$ with constants added to its rows.

An important property of Monge arrays is that of being totally monotone.

***DIST* matrix**

$I_0 = 1$	0	−1	−2	−3	△	△
$I_1 = 2$	−1	−1	−2	−1	−3	△
$I_2 = 3$	−2	0	0	1	−1	−3
$I_3 = 2$	△	−2	−2	0	−2	−2
$I_4 = 1$	△	△	−2	0	−1	−1
$I_5 = 3$	△	△	△	−2	−1	0

***OUT* matrix**

1	0	−1	−2	$-\infty$	$-\infty$
1	1	0	1	−1	$-\infty$
1	3	3	4	2	0
−12	0	0	2	0	0
−13	−13	−1	1	0	0
−14	−14	−14	1	2	3

O_0	O_1	O_2	O_3	O_4	O_5
1	3	3	4	2	3

column numbers

0	**1**	**2**	**3**	**4**	**5**

FIG. 3.2. *The DIST matrix which corresponds to the subsequences "acg" and "ag"; the OUT matrix obtained by adding the values of* I *to the rows of DIST; and the* O *containing the row maxima of OUT. This figure continues Figures* 2.1 *and* 3.1.

DEFINITION 3.4. *A matrix* $M[0 \dots m, 0 \dots n]$ *is* totally monotone *if either condition* 1 *or* 2 *below holds for all* $a, b = 0 \dots m$; $c, d = 0 \dots n$:

1. convex condition: $M[a, c] \geq M[b, c] \Longrightarrow M[a, d] \geq M[b, d]$ *for all* $a < b$ *and* $c < d$;
2. concave condition: $M[a, c] \leq M[b, c] \Longrightarrow M[a, d] \leq M[b, d]$ *for all* $a < b$ *and* $c < d$.

Note that the Monge property implies total monotonicity, but the converse is not true. Therefore, both *DIST* and *OUT* are totally monotone by the concave condition.

Aggarwal et al. [2] gave a recursive algorithm, nicknamed *SMAWK* in the literature, which can compute in $O(n)$ time all row and column maxima of an $n \times n$ totally monotone matrix, by querying only $O(n)$ elements of the array. Hence, one can use *SMAWK* to compute the output row O by querying only $O(n)$ elements of *OUT*. Clearly, if both the full *DIST* and all entries of I are available, then computing an element of *OUT* is O(1) work.

For various solutions to related problems that also utilize Monge and total monotonicity properties, we refer the interested reader to [15], [16], [20], [21], [24], [33], [34], and [41]. In order to efficiently utilize these properties here, we need to address the following two problems:

1. How to efficiently compute *DIST* and represent it in a format which allows direct access to its entries. This will be done in section 3.4.
2. *SMAWK* is intended for a full, rectangular matrix. However, neither *DIST* nor its corresponding *OUT* is rectangular. Since paths in an alignment graph

can assume only a left-to-right, top-to-bottom direction, connections between some input border vertices and some output border vertices are impossible. Therefore, the matrices are missing both a lower-left triangle and upper-right triangle (see Figure 3.2). This problem is addressed in section 3.3.

3.3. Addressing the rectangle problem. The undefined entries of OUT can be complemented in constant time each, as follows:

(a) The missing upper-right triangle entries can be completed by setting the value of any entry $OUT\,[i,j]$ in this triangle to $-\infty$.

(b) Let k denote the maximal absolute value of a score in δ. The missing lower-left triangle entries can be completed by setting the value of any $OUT\,[i,j]$ in this triangle to $-(n+i+1)*k$.

LEMMA 3.5. *Complementing the undefined entries as described above preserves the concave total monotonicity condition of OUT and does not introduce new row-maxima.*

Proof. (a) *Upper right triangle.* All similarity scores in the alignment graph are finite. Therefore, no new column maxima are introduced. Suppose $OUT\,[a,c] \leq OUT\,[b,c]$, $a < b$, and $OUT\,[a,c]$ have been set to $-\infty$. Due to the shape of the redefined upper-right triangle, once a $-\infty$ value in row a is encountered, all future values in row a are also $-\infty$. The future values of row b could either be finite or $-\infty$. Therefore, $OUT\,[a,d] \leq OUT\,[b,d]$ for all $d > c$.

(b) *Lower left triangle.* The worst score appearing in the alignment graph is lower-bounded by $-nk$. Since i is always greater than or equal to zero, the complemented values in the lower-left triangle are upper-bounded by $-(n+1)*k$, and no new column maxima are introduced. Also, for any complemented entry $OUT\,[b,c]$ in the lower-left triangle, $OUT\,[b,c] < OUT\,[a,c]$ for all $a < b$, and therefore the concave total monotonicity condition holds. □

3.4. Incremental update of the new *DIST* information for *G*. In this section we show how to efficiently compute the new *DIST* information for G, using the *DIST* representations previously computed for its prefix blocks plus the information of its new cell.

When processing a new block G, we compute the scores of t new optimal paths, leading from the input border to the new vertex (ℓ_r, ℓ_c) in the new cell of G in its lowest, rightmost corner. These values correspond to column ℓ_c of the *DIST* matrix for G and can be computed as follows.

Entry $[i]$ in column ℓ_c of the *DIST* for G contains the weight of the optimal path from entry i in the input border of G to vertex (ℓ_r, ℓ_c). This path must go through one of the three vertices $(\ell_r - 1, \ell_c)$, $(\ell_r - 1, \ell_c - 1)$, or $(\ell_r, \ell_c - 1)$. Therefore, the weight of the optimal path from entry i in the input border of G to (ℓ_r, ℓ_c) is equal to the maximum among the following three values:

1. Entry $[i]$ of column $\ell_c - 1$ of the *DIST* for the left prefix of G, plus the weight of the horizontal edge leading into (ℓ_r, ℓ_c);
2. Entry $[i]$ of column $\ell_c - 1$ of the *DIST* for the diagonal prefix of G, plus the weight of the diagonal edge leading into (ℓ_r, ℓ_c);
3. Entry $[i]$ of column ℓ_c of the *DIST* for the top prefix of G, plus the weight of the vertical edge leading into (ℓ_r, ℓ_c).

3.4.1. Maintaining direct access to *DIST* columns. In order to compute an entry of OUT in constant time during the execution of *SMAWK*, direct access to *DIST* entries is necessary. This is not straightforward, since, as shown in the previous

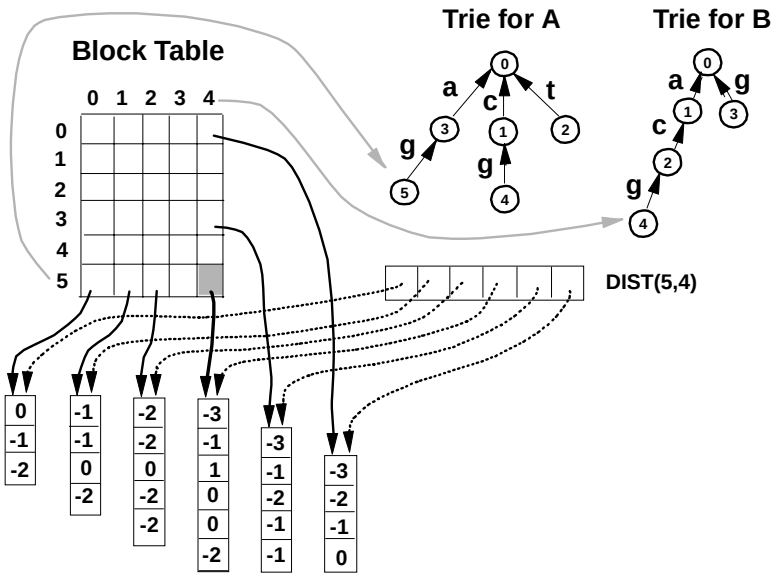


FIG. 3.3. *A table containing an entry for each block of the alignment graph. Entry* (i, j) *of the table represents the block which corresponds to node* i *in the trie for* A *and node* j *in the trie for* B*. The entry for each block in the table points to the start of its new DIST column. Also shown is the vector which contains pointers to all columns of the DIST for block* $(5, 4)$*, as obtained from its ancestor prefix blocks. This figure continues Figures* 2.1*,* 3.1*, and* 3.2*.*

section, for each block only one new *DIST* column has been computed and stored. All other columns besides column ℓ_c of the *DIST* for G need to be obtained from G's prefix ancestor blocks.

Therefore, before the execution of *SMAWK* begins, a vector with pointers to all $t + 1$ columns of the *DIST* for G is constructed (see Figure 3.3). This vector is no longer needed after the computations for G have been completed, and its space can be freed.

The pointers to all columns of the *DIST* for G are assembled as follows. Column ℓ_c is set to the newly constructed vector for G. All columns of indices smaller than ℓ_c are obtained via ℓ_c recursive calls to left prefix blocks of G. All columns of indices greater than ℓ_c are obtained via ℓ_r recursive calls to top prefix blocks of G.

3.4.2. Querying a prefix block and obtaining its *DIST* column in constant time. The *LZ*78 phrases form a trie (see Figure 3.1), and the string to be compressed is encoded as a sequence of names of prefixes of the trie. Each node in the trie contains the serial number of the phrase it represents. Since each block corresponds to a comparison of a phrase from A with a phrase from B, each block will be identified by a pair of numbers, composed of the serial numbers for its corresponding phrases in the tries for A and B.

Another data structure to be constructed is a block table (see Figure 3.3), containing an entry for each partitioned block of the alignment graph. The entry for each block in the table points to the start of its new *DIST* column and can be directly accessed via the block's phrase number index pair.

The left prefix of G can be identified in constant time as a pair of phrase numbers,

the first identical to the serial number of xa, and the second corresponding to the serial number of y, which is the direct ancestor of yb in the trie for B. Similarly, the top prefix of G can be identified in constant time. Given the pair of identification numbers for a block, a pointer to the corresponding *DIST* column can then be obtained directly from the block table.

Time and space analysis. Assuming sequence size n and sequence entropy $h \leq 1$, the $LZ78$ factorization algorithm parses the strings and constructs the tries for A and B in $O(n)$ time. The resulting number of phrases in both A and B is $O(hn/\log n)$. The number of resulting blocks in the alignment graph is equal to the number of phrases in A times number of phrases in B, and is therefore $O(h^2n^2/(\log n)^2)$. For each block G, the following information is computed, in time and space complexity linear with the size of its I/O borders:

1. *Updating the encoding structure for G.* The prefix blocks of G can be accessed in constant time. The vectors of *DIST* column pointers for the prefix blocks have already been freed. However, since each prefix block directly points to its newly computed *DIST* column, all values needed for the computations are still available. Since each entry of the new *DIST* column for G is set to the maximum among up to three sums of pairs, the new *DIST* column for G can be constructed in $O(t)$ time and space.
2. *Maintaining direct access to DIST columns.* Since *prefix* blocks and their *DIST* columns can be accessed in constant time, the vector with pointers to columns of the *DIST* for G can be set in $O(t)$ time.
3. *Propagating I/O values across the block.* Using the information computed for G, and given the I for G obtained from the O vectors for the block above G and the block to its left, the values of O for G are computed via *SMAWK* matrix searching in $O(t)$ time.

Total complexity. Since the work and space for each block is linear with the size of its I/O borders, the total time and space complexity is linear with the total size of the borders of the blocks. The block borders form $O(hn/\log n)$ rows of size $|B|$ each and $O(hn/\log n)$ columns of size $|A|$ each in the alignment graph (see Figure 3.1). Therefore, the total time and space complexity is $O(hn^2/\log n)$.

4. Global similarity optimal alignment trace recovery. The recovery of an optimal global alignment trace between A and B starts at vertex (n, n). The series of block crossing paths is then traced back until vertex $(0, 0)$ is reached. For each block crossed, the internal alignment trace is reported, starting from the output border sink and back to the optimal origin source vertex in the corresponding input border. In order to support the recovery of block-crossing paths in time linear with their size, the computation and storage of the following additional information for a given block G is required:

1. During the propagation stage, for each entry j in the output border of G, the index of the input border entry i, which is the source of the highest scoring path to output border entry j, is saved.
2. During encoding, an additional $O(t)$-sized vector of pointers, the *ancestors* vector, is computed for G. For any output border entry $O[j = 0 \dots t]$, *ancestors*$[j]$ points to the ancestor block of G for which this entry is the new vertex in its new cell. (The value of *ancestors*$[\ell_c]$ is set to G. All columns of indices smaller than ℓ_c are obtained via ℓ_c recursive calls to left prefix blocks of G. All columns of indices greater than ℓ_c are obtained via ℓ_r recursive calls to top prefix blocks of G.)

3. During encoding, G's new vertex (ℓ_r, ℓ_c) is annotated with an additional $O(t)$-sized vector of pointers, denoted *direction*. These pointers are set during the *DIST* column computation described in section 3.4, as follows. The value of *direction*$[i]$ is set according to the direction of the last edge in the optimal path originating at entry i of G's input border and ending at vertex (ℓ_r, ℓ_c).

Given that the optimal path enters through entry j of the output border of G, the trace-back of the part of the path going through G proceeds in two stages. The first stage is a destination and origin initialization stage. This stage includes the fetching of the input row source entry i, which was stored as the origin for the highest scoring path to G's output border entry j (see 1 above). Entry i serves as the destination for the alignment trace-back. In addition, the ancestor prefix block P of G, pointed to by *ancestors*$[j]$ is fetched (see 2 above). The edge recovery begins in block P.

During the second stage, the origin and destination information computed in the first stage is used to trace back the part of the path contained in P, from entry j on P's output border (the new vertex of P) to entry i on its input border. This is done by backtracking through a dynasty of prefix ancestor blocks internal to P, using the *direction* vector computed for each of the traversed blocks (see 3 above). If *direction*$[i]$ of the traversed block specifies a horizontal edge, then the trace-back retreats to the left prefix of P, and an "insertion" operation is reported in the alignment trace. Correspondingly, "substitution" and "deletion" are reported when backtracking to diagonal and top prefix blocks. The recovery continues through a series of prefix blocks of P until the full optimal alignment trace is recovered.

Time and space analysis. The two additional vectors for G, *direction* and *ancestors*, and the input border source entry i, can be computed and stored during the encoding and propagation stages in $O(t)$ time and space.

The work for the first stage in the trace-back can be done in constant time. In the second stage, each edge in the recovered alignment path results in a traversal to a single prefix block. Since prefix blocks and their corresponding direction vectors can be accessed in constant time, a highest scoring global alignment between strings A and B can be recovered in additional time linear in its size, using the $O(hn^2/\log n)$ storage which was allocated during the encoding and propagation stages.

5. Reducing the space complexity. When computing the optimal global alignment value with scoring matrices which follow the "discreteness" condition (see section 1), the *efficient alignment stage algorithm* described in [34] can be extended to support full propagation from the leftmost and upper boundaries to the bottom and rightmost boundaries of G.

This extended propagation algorithm can then be used to compute the values of the global alignment O for G, given the I for G and a minimal encoding of the *DIST* for G. The advantage of this minimal encoding of *DIST* is that, rather than saving an $O(t)$ sized *DIST* column per block, we need to save only a constant number of values per block. The encoding for the new *DIST* column of each block can be computed and stored in constant time and space from the information stored for the left, diagonal, and top prefix blocks of G, using the technique described in section 6 of [48]. This reduces the space complexity to $O(h^2n^2/(\log n)^2)$ while preserving the $O(hn^2/\log n)$ time complexity.

6. The local alignment algorithm.

6.1. Computing the optimal local similarity value. When computing the optimal local similarity value, an optimal path could either be contained entirely in

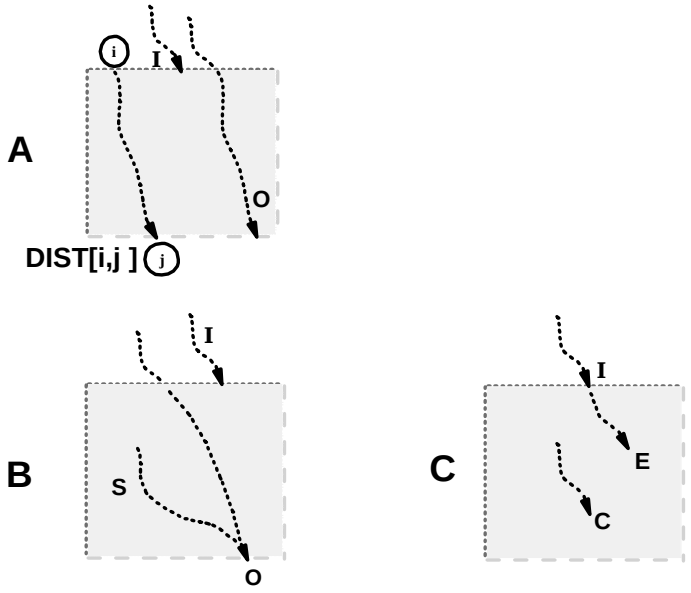


FIG. 6.1. A. *The I/O path weight vectors computed for each block in the global alignment solution. DIST $[i, j]$ will be set to the highest scoring path connecting vertex i in the input border with vertex j in the output border.* B,C. *The vectors of optimal path weights considered for the local alignment computation.*

one block (type C) or be a block-crossing path (see Figure 6.1). A block-crossing path consists of a (possibly empty) S-path, followed by any number of paths leading from the input border of a block to its output border, and ending in an E-path with a highest scoring last vertex. Since an optimal path could begin inside any block, vector O needs to be updated to consider the additional paths originating inside G. Also, since an optimal path could end inside any block, extra bookkeeping is needed in order to keep track of the highest scoring paths ending in each block.

Therefore, in addition to the *DIST* described in section 3, we compute for each block G the following data structures (see Figures 6.1B and 6.1C):

1. E is a vector of size t. $E[i]$ contains the value of the highest scoring path which starts at vertex i of the input border of G and ends inside G. $E[i]$ is computed as the maximum between $E[i]$ for the left prefix of G, $E[i]$ for the top prefix of G, and $DIST\,[i, \ell_c]$.
2. S is a vector of size t. $S[i]$ contains the value of the highest scoring path which starts inside G and ends at vertex i of the output border of G.
   The only new values computed for S are the local alignment scores for the new vertex of G, $S[\ell_c]$. Given the scores $S[\ell_c - 1]$ obtained from the *diagonal prefix*, $S[\ell_c - 1]$ obtained from the *left prefix*, and $S[\ell_c]$ obtained from the *top prefix* of G, as well as the weights of the three edges leading into vertex (ℓ_r, ℓ_c), $S[\ell_c]$ can be computed in $O(1)$ time complexity using the recursion given in section 2.1.
   The values of all other entries of S are then set as follows. The first ℓ_c values of S are copied from the first ℓ_c values of the S computed for the left prefix of G. The last ℓ_r values are copied from the last ℓ_r values of the S vector for the top prefix of G.

3. C is the value of the highest scoring path contained in G, that is, the highest scoring path which originates inside G and ends inside G. C is computed as the maximum between the C value for the left prefix of G, the C value for the top prefix of G, and the newly computed $S[\ell_c]$ as described above.

The S vector computed for G is used to update the values of the output border O, while E and C will be used to compute the weight of the highest scoring path ending in G.

Vector O is first computed from the I and $DIST$ for G as described in section 3.2. At this point, entry $O[i]$ reflects the weight of the optimal path starting anywhere outside G and ending at entry i of the output border. It needs to be updated with the weights of the highest scoring paths starting inside G. This is achieved by resetting $O[i]$ to the maximum between $O[i]$ and $S[i]$.

The weight of the highest scoring path ending in G is computed as $max(Max_{i=0}^{t}\{I[i]+E[i]\},\ C)$.

After the computations for each block have been completed, the overall highest local alignment score for comparing A and B can be computed as the maximum among the values of the highest scoring path ending in each block.

Time and space analysis. Since, as shown in section 3.4.1, each prefix block of G can be accessed in constant time, the values of the S and E vectors for G can be computed and stored in $O(t)$ time and space, and the C value for G can be computed in constant time and space.

Given the S, E, and C vectors for G, the values of O and the weight of the highest scoring path ending in G can be computed in $O(t)$ time each as described above.

The weight of the highest scoring path in the alignment graph can then be computed in an additional $O(h^2n^2/(\log n)^2)$ time as the maximum value among the best values computed for each block.

Since the work and space for each block is linear with the size of its I/O borders, the total time and space complexity of computing the optimal local alignment value is $O(hn^2/\log n)$.

6.2. Optimal alignment trace recovery for the local alignment solution. Similarly to the alignment trace defined in section 4, given a $maxL$ vertex (i_{end}, j_{end}) which was obtained in the previous section, we show how to recover the optimal path ending in this vertex by reporting a trace-back of the edges from vertex (i_{end}, j_{end}) until a start-point vertex (i_{start}, j_{start}) is reached that has value zero.

A block-crossing optimal path consists of a (possibly empty) S-path, followed by any number of paths leading from the input border of a block to its output border and ending in an E-path whose last vertex is (i_{end}, j_{end}).

The recovery starts at vertex (i_{end}, j_{end}) and continues back to the optimal path origin in three stages, as follows:

1. *Recovering the E-path part.* During encoding, whenever the $E[i]$ value of a block is updated by its new vertex, a pointer to the updating block is saved together with the new $E[i]$ value.
   During alignment recovery, given that vertex (i_{end}, j_{end}) ends an $E[i]$ path in G, the corresponding block can be fetched and the path from its new vertex to entry i on its input border recovered, as described in section 4.
2. *Recovering all paths leading from the input border of a block to its output border.* The part of the path contained in each one of these blocks can be recovered as described in section 4.
3. *Recovering the S-path part.* During encoding, when computing the S-score of

the new vertex of each block, the direction of the edge optimizing the score $S[\ell_c]$ of the new vertex of G, denoted $s_{direction}$, is saved with the score.
During the termination of the propagation stage, when setting the score values for each entry in O, a field is set, indicating whether the newly set score value for this entry corresponds to a path originating inside G (an S-path) or a path crossing G. If the score corresponds to an S-path, the recovery of the S-path part utilizes the technique described in section 4, with a slight modification. Instead of the *direction* vector, the $s_{direction}$ field is used for the edge traceback. The recovery halts when an ancestor block is reached whose $S[\ell_c]$ value is zero.

A special case occurs when vertex (i_{end}, j_{end}) is the end point of a C-path. A C-path is, in essence, a halted S-path. During encoding, whenever the C value of a block is updated by its new vertex, a pointer to the updating block is saved together with the new C value. The recovery of the C path in G starts at the new vertex of its corresponding block and continues similarly to the S-path recovery, as described in 3 above.

Time and space analysis. In addition to the values described in section 4, an additional $O(t)$ information (pointers to the $E[i]$ updating blocks) is computed and stored for E-paths, and an additional $O(1)$ information per block is computed and stored for C and S paths. During propagation termination, an additional $O(t)$ information is stored with the O vector.

During recovery, each edge in the recovered alignment path results in a traversal to a single prefix block, for each of the three path parts. Both prefix blocks and their corresponding direction vectors can be accessed in constant time. Therefore, in addition to the basic $O(hn^2/\log n)$ time and space needed for computing the optimal local alignment score $maxL$, an alignment trace ending at a given $maxL$-scoring vertex can be reported in time linear with the size of the trace.

7. Applications to the problem of comparing two run-length encoded strings. A string S is *run-length encoded* if it is described as an ordered sequence of pairs (σ, i), often denoted "σ^i," each consisting of an alphabet symbol σ and an integer i. Each pair corresponds to a *run* in S, consisting of i consecutive occurrences of σ. For example, the string *aabbbbbccc* can be encoded as $a^2b^5c^3$. Such a run-length encoded string can be significantly shorter than the expanded string representation after efficiently encoding the integers (see [14], for example).

Run-length encoding serves as a popular image compression technique, since many classes of images (e.g., binary images in facsimile transmission or for use in optical character recognition) typically contain large patches of identically valued pixels.

Let m and n be the lengths of two run-length encoded strings X and Y, of encoded lengths m' and n', respectively. Previous algorithms for the problem compared two run-length encoded strings using the Levenshtein edit distance [36] and the LCS similarity measure [26]. For the LCS metric, Bunke and Csirik [9] presented an $O(mn' + nm')$ time algorithm, while Apostolico, Landau, and Skiena [6] described an $O(m'n' \log(m'n'))$ time algorithm. Mitchell [42] has obtained an $O((d + m' + n') \log(d + m' + n'))$ time algorithm for a more general string matching problem in run-length encoded strings, where d is the number of matches of compressed characters. Both Arbell, Landau, and Mitchell [1] and Mäkinen, Navarro, and Ukkonen [38] independently obtained an $O(m'n + n'm)$ time algorithm for computing the edit distance between two run-length encoded strings for the Levenshtein distance metric.

Mäkinen, Navarro, and Ukkonen [38] posed as an open problem the challenge of

extending these results to more general scoring schemes, since in those applications which are related to image compression, the change from a pixel value to the next is smooth. Here, we will show how to extend the results to apply them to any distance or similarity scoring scheme with additive gap scores.

In this solution, the alignment graph is also partitioned into blocks. But rather than using the $LZ78$ partition described in section 3.1, each block here consists of two runs—one of X and one of Y. This results in the partition of the alignment graph into $m'n'$ blocks. The algorithm suggested also propagates accumulated scores from the left and upper boundaries of each block to its bottom and right boundaries.

Consider the block R for comparing the run α^i of X with the run β^j of Y. An edge in R could be assigned one of three possible weight values: D(diagonal), H(horizontal), or V(vertical).

Let Δ_h and Δ_w denote the difference in row index values and column index values, respectively, between entry i on the input border of R and entry j on the output border of R.

We show how to compute $DIST\,[i,j]$ (which is the cost of the best scoring path from entry i in the input border of the block to entry j in the output border of the block) in constant time, given Δ_h and Δ_w for the input and output entries, and the values D, H, and V.

- $H + V \leq D$. Clearly, an optimal path from i to j can use all possible diagonal edges and only then the minimal number of remaining H and V edges necessary to reach j.
  Therefore, $DIST\,[i,j]$ obtains one of three values:
  1. If $\Delta_w = \Delta_h$, then $DIST\,[i,j] = D \times \Delta_h$.
  2. If $\Delta_w > \Delta_h$, then $DIST\,[i,j] = D \times \Delta_h + H \times (\Delta_w - \Delta_h)$.
  3. If $\Delta_w < \Delta_h$, then $DIST\,[i,j] = D \times \Delta_w + V \times (\Delta_h - \Delta_w)$.
- $H + V > D$. In this case, an optimal path never uses any diagonal edge. The path includes only the minimal number of H edges, and the minimal number of V edges necessary to reach j from i. In this case, $DIST\,[i,j] = H \times \Delta_w + V \times \Delta_h$.

Therefore, $DIST\,[i,j]$ can be easily computed in constant time when using the general scoring scheme described in section 2.1.

Time and space analysis. The O vector for each block is computed using *SMAWK*. Vector I for block R can easily be obtained from the O vectors for the block above R and the block to its left, in time linear with the sides of R. The "rectangle" problem can be solved as in section 3.2. Therefore, any value $OUT\,[i,j] = I[i] + DIST\,[i,j]$ can be computed in constant time.

Since the work for each block is linear with the size of its I/O borders, the total time complexity is linear with the total size of the borders of the blocks, which is $O(m'n + n'm)$.

Note that alternative methods for achieving linear-time propagation across run-length compressed blocks can be obtained by adapting any of the queue algorithms described in [16], [20], [27], and [34].

Since all relevant $DIST$ entry values are computed "on the fly" and do not need to be stored, Hirschberg's method [26] can be applied to achieve an algorithm with a space complexity that is linear with the size of the uncompressed strings.

Open problems. The subquadratic sequence comparison algorithms presented in this paper are perhaps close to optimal in time complexity. However, an important concern is the space complexity of the algorithms. If only the similarity score

value is required, the classical, quadratic time sequence alignment algorithm can easily be implemented to run in linear space by keeping only two rows of the dynamic programming table alive at each step. If the recovery of either global or local optimal alignment traces is required, quadratic-time and linear-space algorithms can be obtained by applying Hirschberg's refinement to the classical sequence alignment algorithms [10], [26], [28]. We post as an open problem the challenge of further reducing the space requirement of the algorithms described in this paper without impairing their subquadratic time complexity.

Acknowledgment. We are grateful to Dan Gusfield for a helpful discussion.

REFERENCES

[1] O. Arbell, G. M. Landau, and J. Mitchell, *Edit distance of run-length encoded strings*, Inform. Process. Lett., to appear.

[2] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber, *Geometric applications of a matrix-searching algorithm*, Algorithmica, 2 (1987), pp. 195–208.

[3] A. Aggarwal and J. Park, *Notes on searching in multidimensional monotone arrays*, in Proceedings of the 29th IEEE Symposium on Foundations of Computer Science, White Plains, NY, 1988, pp. 497–512.

[4] A. Amir, G. Benson, and M. Farach, *Let sleeping files lie: Pattern matching in Z-compressed files*, J. Comp. Sys. Sci., 52 (1996), pp. 299–307.

[5] A. Apostolico, M. J. Atallah, L. L. Larmore, and S. McFaddin, *Efficient parallel algorithms for string editing and related problems*, SIAM J. Comput., 19 (1990), pp. 968–988.

[6] A. Apostolico, G. M. Landau, and S. Skiena, *Matching for run length encoded strings*, J. Complexity, 15 (1999), pp. 4–16.

[7] T. C. Bell, J. C. Cleary, and I. H. Witten, *Text Compression*, Prentice–Hall, Englewood Cliffs, NJ, 1990.

[8] G. Benson, *A space efficient algorithm for finding the best nonoverlapping alignment score*, Theoret. Comput. Sci., 145 (1995), pp. 357–369.

[9] H. Bunke and J. Csirik, *An improved algorithm for computing the edit distance of run length coded strings*, Inform. Process. Lett., 54 (1995), pp. 93–96.

[10] K. M. Chao, R. Hardison, and W. Miller, *Recent developments in linear-space alignment methods: A mini survey*, J. Comp. Biol., 1 (1994), pp. 271–291.

[11] M. Crochemore, G. M. Landau, and M. Ziv-Ukelson, *A sub-quadratic sequence alignment algorithm for unrestricted cost matrices*, in Proceedings of the 13th ACM-SIAM Symposium On Discrete Algorithms, San Francisco, 2002, pp. 679–688.

[12] M. Crochemore, *Transducers and repetitions*, Theoret. Comput. Sci., 45 (1986), pp. 63–86.

[13] M. Crochemore and W. Rytter, *Text Algorithms*, Oxford University Press, London, 1994.

[14] P. Elias, *Universal codeword sets and representation of integers*, IEEE Trans. Inform. Theory, 21 (1975), pp. 194–203.

[15] D. Eppstein, *Sequence comparison with mixed convex and concave costs*, J. Algorithms, 11 (1990), pp. 85–101.

[16] D. Eppstein, Z. Galil, and R. Giancarlo, *Speeding up dynamic programming*, in Proceedings of the 29th IEEE Symposium on Foundations of Computer Science, White Plains, NY, 1988, pp. 488–296.

[17] D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano, *Sparse dynamic programming* I*: Linear cost functions*, J. ACM, 39 (1992), pp. 546–567.

[18] D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano, *Sparse dynamic programming* II*: Convex and concave cost functions*, J. ACM, 39 (1992), pp. 568–599.

[19] M. Farach and M. Thorup, *String matching in Lempel–Ziv compressed strings*, Algorithmica, 20 (1998), pp. 388–404.

[20] Z. Galil and R. Giancarlo, *Speeding up dynamic programming with applications to molecular biology*, Theoret. Comput. Sci., 64 (1989), pp. 107–118.

[21] Z. Galil and K. Park, *A linear-time algorithm for concave one-dimensional dynamic programming*, Inform. Process. Lett., 33 (1990), pp. 309–311.

[22] L. Gasieniec, M. Karpinski, W. Plandowski, and W. Rytter, *Randomised efficient algorithms for compressed strings: The finger-print approach*, in Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching, Laguna Beach, CA, 1996, Lecture Notes in Comput. Sci. 1075, Springer-Verlag, New York, 1996, pp. 39–49.

[23] L. Gasieniec and W. Rytter, *Almost optimal fully LZW compressed pattern matching*, in proceedings of the Data Compression Conference, J. Storer, ed., 1999.
[24] R. Giancarlo, *Dynamic programming: Special cases*, in Pattern Matching Algorithms, A. Apostolico and Z. Galil, eds., Oxford University Press, London, 1997, pp. 201–232.
[25] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, Cambridge, UK, 1997.
[26] D. S. Hirschberg, *A linear space algorithm for computing maximal common subsequences*, Comm. ACM, 18 (1975), pp. 341–343.
[27] D. S. Hirshberg and L. L. Larmore, *The least weight subsequence problem*, SIAM J. Comput., 16 (1987), pp. 628–638.
[28] X. Huang and W. Miller, *A time-efficient, linear space local similarity algorithm*, Adv. Appl. Math., 12 (1991), pp. 337–357.
[29] S. K. Kannan and E. W. Myers, *An algorithm for locating nonoverlapping regions of maximum alignment score*, SIAM J. Comput., 25 (1996), pp. 648–662.
[30] J. Karkkainen, G. Navarro, and E. Ukkonen, *Approximate string matching over Ziv–Lempel compressed text*, in Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching, Montreal, 2000, Lecture Notes in Comput. Sci. 1848, Springer-Verlag, New York, 2000, pp. 195–209.
[31] J. Karkkainen and E. Ukkonen, *Lempel–Ziv parsing and sublinear-size index structures for string matching*, in Proceedings of the Third South American Workshop on String Processing (WSP '96), Recife, Brazil, 1996, N. Ziviani, R. Baeza-Yates, and K. Guimaraes, eds., Carleton University Press, Montreal, 1996, pp. 141–155.
[32] T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, and S. Arikawa, *Shift-And approach to pattern matching in LZW compressed text*, in Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching, Jerusalem, 1999, Lecture Notes in Comput. Sci. 1645, Springer-Verlag, New York, 1999, pp. 1–13.
[33] M. M. Klawe and D. J. Kleitman, *An almost linear time algorithm for generalized matrix searching*, SIAM J. Discrete Math., 3 (1990), pp. 81–97.
[34] G. M. Landau and M. Ziv-Ukelson, *On the common substring alignment problem*, J. Algorithms, 41 (2001), pp. 338–359.
[35] A. Lempel and J. Ziv, *On the complexity of finite sequences*, IEEE Trans. Inform. Theory, 22 (1976), pp. 75–81.
[36] V. I. Levenshtein, *Binary codes capable of correcting, deletions, insertions and reversals*, Soviet Phys. Dokl, 10 (1966), pp. 707–710.
[37] M. G. Main and R. J. Lorentz, *An $O(n \log n)$ algorithm for finding all repetitions in a string*, J. Algorithms, 5 (1984), pp. 422–432.
[38] V. Mäkinen, G. Navarro, and E. Ukkonen, *Approximate matching of run-length compressed strings*, in Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching, Jerusalem, 2001, Lecture Notes in Comput. Sci. 1645, Springer-Verlag, New York, 2001, pp. 1–13.
[39] U. Manber, *A text compression scheme that allows fast searching directly in the compressed file*, in Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, Asilomar, CA, 2001, Lecture Notes in Comput. Sci. 807, Springer-Verlag, New York, 2001, pp. 31–49.
[40] W. J. Masek and M. S. Paterson, *A faster algorithm for computing string edit distances*, J. Comput. Systems Sci., 20 (1980), pp. 18–31.
[41] W. Miller and E. W. Myers, *Sequence comparison with concave weighting functions*, Bull. Math. Biol., 50 (1988), pp. 97–120.
[42] J. Mitchell, *A Geometric Shortest Path Problem, with Application to Computing a Longest Common Subsequence in Run-Length Encoded Strings*, Technical report, Department of Applied Mathematics, SUNY Stony Brook, 1997.
[43] G. Monge, *Déblai et Remblai*, Mémoires de l'Academie des Sciences, Paris, 1781.
[44] G. Navarro, T. Kida, M. Takeda, A. Shinohara, and S. Arikawa, *Faster approximate string matching over compressed text*, Proceedings of the Data Compression Conference (DCC2001), 2001, IEEE Computer Society, Snowbird, UT, pp. 459–468.
[45] G. Navarro and M. Raffinot, *A general practical approach to pattern matching over Ziv–Lempel compressed text*, in Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching, Warwick, UK, 1999, Lecture Notes in Comput. Sci. 1645, Springer-Verlag, New York, 1999, pp. 14–36.
[46] G. Navarro and M. Raffinot, *Boyer-Moore string matching over Ziv-Lempel compressed text*, in Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching, Montreal, 2000, Lecture Notes in Comput. Sci. 1848, Springer-Verlag, New York, 2000, pp. 166–180.

[47] D. Sankoff and J. B. Kruskal, eds., *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison–Wesley, Reading, MA, 1983.

[48] J. P. Schmidt, *All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings*, SIAM J. Comput., 27 (1998), pp. 972–992.

[49] Y. Shabita, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa, *Speeding up pattern matching by text compression*, in Proceedings of the 4th Italian Conference on Algorithms and Complexity (CIAC 2000), Rome, Lecture Notes in Comput. Sci. 1767, Springer-Verlag, New York, 2000, pp. 306–315.

[50] T. F. Smith and M. S. Waterman, *Identification of common molecular subsequences*, J. Molec. Biol., 147 (1981), pp. 195–197.

[51] W. Szpankowski and P. Jacquet, *Asymptotic behavior of the Lempel-Ziv parsing scheme and digital search trees*, Theoret. Comput. Sci., 144 (1995), pp. 161–197.

[52] T. A. Welch, *A technique for high performance data compression*, IEEE Trans. Comput., 17 (1984), pp. 8–19.

[53] J. Ziv and A. Lempel, *A universal algorithm for sequential data compression*, IEEE Trans. Inform. Theory, 23 (1977), pp. 337–343.

[54] J. Ziv and A. Lempel, *Compression of individual sequences via variable rate coding*, IEEE Trans. Inform. Theory, 24 (1978), pp. 530–536.